الجمهورية العربية السورية
جامعة حلب
كلية الهندسة الكهربائية والإلكترونية
قسم هندسة الحواسيب

# تصميم مترجم بين لغة C تحت نظام تشغيل VXWORKS ولغة IF(INTERMEDIATE FORMULA) للتحقق من صحة البرامج

رسالة أعدت لنيل درجة الماجستير في هندسة الحواسيب

إعداد

المهندسة مروة دحدوح

٢٠٠٨-٢٠٠٩

الجمهورية العربية السورية
جامعة حلب
كلية الهندسة الكهربائية والإلكترونية
قسم هندسة الحواسيب

# تصميم مترجم بين لغة C تحت نظام تشغيل VXWORKS ولغة IF(INTERMEDIATE FORMULA) للتحقق من صحة البرامج

رسالة أعدت لنيل درجة الماجستير في هندسة الحواسيب

إعداد
المهندسة: مروة دحدوح

بإشراف

الدكتور سهيل خواتمي
أستاذ مساعدفي قسم الشبكات الحاسوبية
كلية الهندسة المعلوماتية

الدكتور أحمد بدر الدين الخضر
مدرس في قسم هندسة الحواسيب
كلية الهندسة الكهربائية والإلكترونية

**تاريخ مناقشة الرسالة 14/10/2009**

قدمت هذه الرسالةاستكمالاً لمتطلبات نيل درجة الماجستير في الهندسة المعلوماتية في قسم هندسة الحواسيب من كلية الهندسة الكهربائية والإلكترونية بجامعة حلب.

Submitted in partial fulfillment of the requirements of master Degree of computer engineering at the faculty of Electricity and Electronic Engineering, University of Aleppo.

**الجمهورية العربية السورية**

**جامعة حلب**

# شـهادة

نشهد أن العمل الموصوف في هذه الرسالة هو نتيجة بحث علمي قامت به المرشحة المهندسة مروة دحدوح تحت إشراف الدكتور المهندس سهيل خواتمي في قسم النظم والشبكات الحاسوبية من كلية الهندسة المعلوماتية بجامعة حلب وبالتعاون مع الدكتور المهندس أحمد بدر الدين الخضر في قسم هندسة الحواسيب من كلية للهندسة الكهربائية و الإلكترونية بجامعة حلب، وأنّ أي رجوع لمراجع ذكرت في هذا العمل موثقة في نص الرسالة.

| المرشحة | المشرف الرئيسي | المشرف المساعد |
|---|---|---|
| المهندسة مروة دحدوح | الدكتور سهيل خواتمي | الدكتور أحمد بدر الدين الخضر |

# Certification

It is Herby certified that the work described in this thesis is the result of the author's own investigation under the main supervision of Dr. Souheil Khawatmi – Department of Computer Engineering, Faculty of Electrical and Electronic Engineering, University of Aleppo and the assistant supervisor Dr.A.B.Alkhoder – Department of computer Engineering, Faculty of Electricity and electronic engineering, Aleppo university.

Any references of other research work have been duty acknowledged in the text.

| Candidate | Main Supervisor | Assistant Supervisor |
|---|---|---|
| Eng. Marwa Dahdouh | Dr. Souheil Khawatmi | Dr. A.B. Alkhoder |

# تصريح

أصرح بأن هذا البحث "**تصميم مترجم بين لغة C تحت نظام تشغيل VxWorks ولغة IF(Intermediate Formula) للتحقق من صحة البرامج**" لم يسبق أن قبل لأي شهادة ولا هو مقدم حالياً للحصول على شهادة أخرى.

المرشحة

**م. مروة دحدوح**

# Declaration

It is Herby declared that this work has not already been accepted for any degree, and it is not being submitted concurrently for any other degree.

Candidate

M.Dahdouh

# شهادة لجنة الحكم والمناقشة

أُنجزت هذه الرسالة وهي بعنوان "**تصميم مترجم بين لغة C تحت نظام تشغيل VxWorks ولغة IF(Intermediate Formula) للتحقق من صحة البرامج**" ونوقشت بتاريخ

من قبل الدكاترة:

**الدكتور سهيل خواتمي**
أستاذ مساعد في قسم الشبكات
كلية الهندسة المعلوماتية
جامعة حلب

**الدكتور أحمد صقر أحمد**
أستاذ في قسم الشبكات
كلية الهندسة المعلوماتية
جامعة تشرين

**الدكتور عامر بوشي**
مدرس في قسم هندسة الحواسيب
كلية الهندسة الكهربائية والإلكترونية
جامعة حلب

بسم الله الرحمن الرحيم

قل رب زدني علما

# كلمة شكر

إلى كل من نقل إلينا العلم والمعرفة بصدق و إخلاص ومد لنا يد العون والنجاة..
إلى من قادنا إلى طريق النور والصواب وأزال عنا غشاوة الجهل والظلام ...
الدكاترة الكرام في كلية الهندسة المعلوماتية وكلية الهندسة الكهربائية والإلكترونية.

أخص بالشكر الدكتور سهيل خواتمي الذي لم يتوان عن تقديم كل مساعدة علمية ممكنة لي والدكتور أحمد بدر الدين الخضر الذي منحني الكثير من وقته وعلمه إلى أن اكتمل بحثي ور أى النور .

إلى كل من له فضل في إنجاز عملي هذا.. لساني عاجز عن الشكر لما قدمتموه لي فلكم مني كل الاحترام و التقدير والامتنان على مدى الحياة ...

# إهداء

إلى النور الذي يضيء لي درب الحياة بالعزم والثبات.. قدوتي ومثلي الأعلى ...

أبي الغالي

إلى الملاك الطاهر التي كان دعاؤها بلسماً للجراح وحافزاً للنجاح ...

أمي الغالية

إلى توءم روحي.. من شجعتني وساندتني للمضي قدماً نحو الأمام ...

أخواتي الأحبة

أزهار... غصون

إلى من شاركني دروب الحياة وخالط شجوني وعايش أفراحي...

إخوتي الأحبة

زاهر ... علي ... عبد اللطيف

# منهجية البحث

مر هذا البحث بالمراحل التالية:

**المرحلة الأولى: دراسة تفصيلية للغتي البرمجة C/VxWorks وIF.**

تم خلال هذه المرحلة القيام بدراسة نظرية شاملة لكافة اللغات والأدوات البرمجية التي اعتُمدت لتحقيق الهدف المطلوب من هذا البحث. شملت هذه الدراسة النقاط التالية:

- أنظمة الزمن الحقيقي والشروط الزمنية التي تُقيّد تنفيذها والبيئات المختلفة المرتبطة بها.
- لغة C/VxWorks(التي تُمثّل لغة دخل المترجم).
- لغة IF (لغة الخرج).
- الأدوات واللغات البرمجية التي تساعد في التحقق من صحة الشيفرة البرمجية خلال مرحلة التحقيق النهائي.

**المرحلة الثانية: بناء المترجم المنشود.**

تم خلال هذه المرحلة القيام ببناء المترجم باعتماد تقنيات صناعة المترجمات المختلفة حيث تم إنجاز مرحلتي التحليل اللفظي والتركيبي وكتابة قواعد لغة الدخل وفق طريقة BNF(Backus-Naur Form) وتحديد التعليمات المتقابلة بين دخل المترجم (لغة C/VxWorks) وخرجه (لغة IF) بهدف توليد الشيفرة البرمجية النهائية كما تم أيضاً ضمن هذه المرحلة اختبار المترجم النهائي الناتج والتأكد من صحة نتائجه.

**المرحلة الثالثة: التحقق من صحة القيود البرمجية.**

تم خلال هذه المرحلة اختصار أتومات ملف المحاكاة الشامل المقابل لبرنامج لغة IF ومعالجته بتطبيق خوارزميات خاصة تساعد في تمييز مسار القيد الزمني المراد التحقق من صحته بشكل مباشر.

**المرحلة الرابعة: دراسة تطبيقات عملية شاملة.**

خلال هذه المرحلة قمنا بتنفيذ عدد من التطبيقات العملية للتأكد من صحة نتائج البحث واختبار مراحله المختلفة وقد تم ذلك باستخدام مترجمنا في ترجمة تعليمات تطبيق عملي يتناول نظام الحقيبة الهوائية ويدرس القيود الزمنية المفروضة عليها مع السيناريوهات المختلفة الممكنة، كما تم اختبار البحث وفق تطبيق آخر يتناول عمل قارئ السيدي من وجهة نظر الزمن الحقيقي للتأكد من كفاءة المترجم وأدائه العالي وشمولية تعليماته.

# المنشورات

قامت الطالبة بنشر بحث علمي في **مجلة بحوث جامعة حلب** عام ٢٠٠٩ من مجلة بحوث جامعة حلب – سلسلة العلوم الهندسة وهو بعنوان:

**"تصميم مترجم يساعد في التحقق من صحة تطبيقات نظم الزمن الحقيقي"**

# الفهرس


**الفصل الأول: مقدمة ........................................................................ ١**

1.1 تمهيد ........................................................................ 2

2.1 مجال اهتمام البحث ........................................................................ 3

3.1 أهمية البحث وأهدافه ........................................................................ 4

4.1 خطوات البحث: ........................................................................ 4

5.1 منهجية البحث ........................................................................ 5

6.1 الخطوط العامة للأطروحة ........................................................................ 7

7.1 الخلاصة ........................................................................ 8

**الفصل الثاني: أنظمة الزمن الحقيقي وأدواته ........................................................................ 9**

1.2 مقدمة ........................................................................ 10

2.2 تعريف أنظمة الزمن الحقيقي ........................................................................ 10

3.2 بناء نظام زمن حقيقي ........................................................................ 12

4.2 نظام التشغيل VxWorks ........................................................................ 12

5.2 عناصر الزمن الحقيقي وبرمجتها في نظام VxWorks ........................................................................ 14

6.2 مراحل تطوير نظام الزمن الحقيقي ........................................................................ 16

7.2 لغة IF (Intermediate Formula language) ........................................................................ 18

1.7.2 الهيكلية البرمجية العامة للغة IF ........................................................................ 20

2.7.2 تقنيات برمجية للتحقق من الصحة ........................................................................ 23

8.2 أدوات برمجية داعمة ........................................................................ 25

1.8.2 أداة LACATRE ........................................................................ 26

2.8.2 أداة Cygwin ........................................................................ 26

9.2 الخاتمة ........................................................................ 26

**الفصل الثالث: المترجمات وأدواتها ........................................................................ 27**

1.3 مقدمة ........................................................................ 28

2.3 تعريف المترجم ........................................................................ 29

3.3 مراحل بناء المترجم ..................................................................... 29
4.3 بناء المحلل اللفظي والتركيبي ............................................................. 31
1.4.3 قواعد BNF للغة C/VxWorks................................................... 31
2.4.3 تقنية (Lex (A Lexical Analyzer Generator ...................................... 32
3.4.3 تقنية (Yacc (Yet Another Compiler - Compiler ................................ 33
4.4.3 ملف التحليل التنفيذي النهائي ................................................... 34
5.3 الخاتمة........................................................................................ 36

**الفصل الرابع: C/VxWorks to IF ........................................................ 37**
1.4 مقدمة ....................................................................................... 38
2.4 المقابلة بين عناصر C/Vxworks وIF .................................................. 38
1.2.4 المهمة Task.................................................................... 39
2.2.4 المؤقت (WatchDog (alarm.................................................... 41
3.2.4 السيمافور الثنائي Binary Semaphore ......................................... 42
4.2.4 السيمافور التعدادي Counter Semaphore .................................... 44
5.2.4 سيمافور الاستبعاد المتبادل Mutual Exclusion Semaphore..................... 45
6.2.4 صندوق البريد (رتل الرسائل) MessageQueue.................................. 45
7.2.4 برنامج خدمة المقاطعة (ISR (Interrupt Service Routine ......................... 47
3.4 القواعد المتبعة في عملية المقابلة ومعالجة المشاكل البرمجية ..................... 49
1.3.4 قواعد تسمية المتحولات وأنواع بياناتها ........................................ 50
2.3.4 مقابلة التعليمات البرمجية الأساسية للغة C ضمن نظام VxWorks ............... 50
3.3.4 مشاكل في البنية الهيكلية القواعدية للغة IF ................................... 52
4.4 اختبار تكاملية المترجم.................................................................. 55
5.4 الخاتمة........................................................................................ 62

**الفصل الخامس: التحقق من القيود الزمنية................................................ 63**
1.5 مقدمة ....................................................................................... 64
2.5 المعالجة البرمجية للأتومات............................................................. 64
1.2.5 خوارزمية الاختصار ............................................................... 65
2.2.5 تحسين مناقلات الأتومات المختصر ............................................. 70
3.2.5 خوارزمية إعادة التسمية........................................................ 72

3.5 الخاتمة.......................................................................................................74

**الفصل السادس: تطبيقات عملية شاملة ............................................................................75**
1.6 مقدمة .........................................................................................................76
2.6 تطبيقات توضيحية ..............................................................................................76
1.2.6 عمل السيمافور الثنائي..........................................................................................76
2.2.6 عمل المؤقت الزمني ...........................................................................................82
3.2.6 عمل علبة البريد ...............................................................................................85
3.6 تطبيقات شاملة للتحقق من صحة القيود الزمنية ....................................................................87
1.3.6 تطبيق الحقيبة الهوائية .........................................................................................87
1.1.3.6 حالة رجوع الكرسي نحو الخلف..............................................................................94
2.1.3.6 حالة فتح الحقيبة الهوائية ...................................................................................97
3.1.3.6 حالة السرعة بطيئة لا تتطلب أي استجابة ......................................................................99
2.3.6 تطبيق قارئ السيدي.............................................................................................100
1.2.3.6 السواقة تتجه نحو الأمام.....................................................................................104
2.2.3.6 السواقة تتجه نحو الخلف ....................................................................................106
4.6 مناقشة النتائج....................................................................................................107
5.6 الخاتمة..........................................................................................................108

**الفصل السابع: الملخص ومناقشة النتائج..................................................................................109**
1.7 الملخص.........................................................................................................110
2.7 النتائج..........................................................................................................111
3.7 آفاق مستقبلية للتطوير ............................................................................................111

**الملحق /I/: قواعد لغة C/VxWorks مصاغة وفق BNF.............................................................114**

**الملحق /II/: المقابلة بين أهم توابع لغة C/VxWorks ولغة IF ..........................................118**

**الملحق /III/: واجهات التطبيق ............................................................................................123**

**الملحق /IV/: الشيفرة البرمجية المقابلة لخوارزميات المعالجة..........................................................130**

**المراجع...........................................................................................................................136**

# ١

# الفصل الأول

# مقدمة

1.1 تمهيد.

2.1 مجال اهتمام البحث.

3.1 أهمية البحث وأهدافه.

4.1 خطوات البحث.

5.1 منهجية البحث.

6.1 الخطوط العامة للأطروحة.

7.1 الخاتمة.

# مقدمة

## 1.1 تمهيد:

"من الحكمة أن تنظر دوماً إلى أمامك لكن من الصعوبة أن تنظر إلى أكثر مما تستطيع أن ترى" مبدأ صحيح لقرار صناعة البرمجيات، تأتي محاولة بناء أي نظام برمجي نتيجة الحاجة إلى نظام معلومات جديد يؤدي هدفاً معيناً يتطلبه الواقع الملموس وليس استجابة لتوقعات وتخمينات لحاجات مستقبلية مجهولة قد لا تحدث، كما أن تطوير أي برمجية هو ترجمة منطقية لمتطلبات المستخدم الذي يطرح هذه الأفكار التطويرية ويبلورها بما يساعده على الإنجاز الأمثلي للعمل المسند إليه.

لاستيعاب تطوير النظم البرمجية، لابد أن ندرك أن للنظام البرمجي دورة حياة كاملة تماماً مثل أي نظام حي أو إنتاج منتج جديد، فالنظام هو مجموعة مُركّبة لعناصر متكافلة مربوطة مع بعضها البعض وفقاً لخطة متكاملة لإنجاز غاية محددة وتُعد مرحلة تحليل النظم وتصميمها هي المفتاح الفعّال إلى حياة نظام برمجي يستنبط الحاجات والموارد الدفينة ويترجمها إلى عمل قابل للنمو، كما أن مقارنة المواصفات المطلوبة للهدف المنشود مع ما تم إنجازه في كل مرحلة من مراحل العمل وتحديد مدى مطابقته يمنحنا سهولةً في التعديل وتداركاً سريعاً للأخطاء المرتكبة أكثر مما لو تم اكتشاف هذه التجاوزات الخاطئة بعد إتمام العمل كاملاً الأمر الذي يقتضي إعادة إنجاز كافة الخطوات اللاحقة وبذل الجهد والوقت المضاعف.

لا يوجد تصميم لبرنامج أو نظام محكم تماماً، فالاتصالات بين المستخدم والمُصمم ليست واضحة أو كاملة دوماً كما أن ضيق الوقت عادة ما يؤدي إلى مشاكل في تحليل البرمجية وتصميمها وقد يُتبع ذلك بإهمال المبرمجين كتابتهم للشيفرة، فإذا ما أُضيف بُعد زمني للنظام أصبح الوضع أكثر تعقيداً وتداخلاً وظهرت قيود إضافية للعمل لابد من الالتزام التام بها لِيُقال إن برمجيتنا قابلة للتطبيق والاستثمار، وهذا قد يُنشئ أخطاء جسيمة في النظام تؤخر جاهزيته وقابلية استخدامه.

لا أحد يرغب في الفشل، خاصة في مشروع ضخم وحرج يحترم الجزئيات الزمنية للعمل مثل بناء نظام زمن حقيقي، يمكن أن ينتج الفشل في هذه الأنظمة في أحسن الأحوال عن تبديد العائد على الاستثمار وتأخيره نتيجة عدم إمكانية الاختبار المباشر لصحة برمجية الزمن الحقيقي لعدم توفر الأدوات البرمجية المتكاملة المساعدة في ذلك، وفي أحسن الأحوال تُقاد المؤسسة التي تتعهد بإنجاز العمل إلى الإفلاس أو تهديد مجرى

الوظائف الرئيسية لفريق عملها ما قد يُسبب ذلك خسائر مادية وبشرية خطيرة. لكي تكون نظم الزمن الحقيقي قابلة للتطبيق السليم يجب استنباط الضوابط الصحيحة التي تحكمها وتُحدد درجة صحتها ومطابقتها للهدف المنشود وهذا يتطلب اختباراً حيوياً متكاملاً والعمل وفق الافتراض المنطقي القائل إذا كانت أجزاء النظام صحيحة فلسوف يتم تحقيق الهدف البرمجي بنجاح.

## 2.1 مجال اهتمام البحث:

يركز هذا البحث على أنظمة الزمن الحقيقي ذات القيود الزمنية المختلفة ويسعى إلى إيجاد أسلوب علمي منطقي صحيح للقيام بعملية التحقق من صحة هذه القيود المفروضة على النظام قبل وضعه في مرحلة الاستثمار النهائي إذ أن أي تجاوزات أو إهمال للقيود الزمنية قد يؤدي إلى انهيار كامل النظام أو إتلافه وربما قد يهدد حياة البشر المحيطين بذاك النظام.

تكمن أهمية هذا البحث في أنه يقدّم أداة برمجية خاصة بأنظمة الزمن الحقيقي وهي عبارة عن مترجم خاص بقواعد لغة C التي تدعم كتابة البرامج ضمن نظام التشغيل VxWorks إضافةً إلى معالجة التعليمات الخاصة بالزمن الحقيقي، إذ أن كل نظام تشغيل يعتمد في توصيف مهامه على لغة برمجية معينة تساعده في برمجة الوظائف التي يُتيحها للمستخدم ونظام VxWorks يعتمد في ذلك على لغة C التي تُقدّم له مجموعة من التعليمات والبروتوكولات والأدوات على شكل كتل وواجهة برمجية تطبيقية API(Application-Programming Interface) يمكن للمبرمج استخدامها مباشرة في تطوير برامجه ضمن نظم الزمن الحقيقي وفي التعامل مع أغراض التزامن المختلفة (رسائل، إشارات، سيمافور وغيرها). رغم أن مجال بناء المترجمات مطروق كثيراً إذ غالباً ما نسمع عن بناء مترجم جديد للغة برمجة ما إلا أن صناعة مترجم لأنظمة الزمن الحقيقي يُعتبر موضوعاً جديداً لم تتناوله كافة الدراسات السابقة، ومن هنا تأتي أهمية مترجمنا هذا الذي يقوم بتحليل برامج C/VxWorks ذات عناصر الزمن الحقيقي المختلفة ومقابلتها وتحويلها إلى تعليمات تتوافق مع لغة برمجية أخرى تدعم التحقق من صحة هذه الأنظمة هي لغة IF (Intermediate Formula Language) التي يتم وفقها توصيف النظام على شكل عدة عمليات (processes) مختلفة مع حالاتها والإشارات المتبادلة بينها، إذ أننا وفق لغة C/VxWorks نستطيع كتابة التطبيقات البرمجية فقط دون إمكانية التحقق من صحة الشيفرة البرمجية وتأديتها الهدف البرمجي المنشود من هذه التطبيقات قبل وضعها في مرحلة الاستثمار النهائي.

يتم وفق بحثنا أيضاً معالجة برنامج النظام المكتوب بلغة IF الناتج عن مترجمنا من خلال استخدام إحدى الأدوات البرمجية المتاحة من قبل هذه اللغة بأسلوب يُمكّننا من الحصول على مخطط التنفيذ الشامل

المقابل للنظام المدروس والمتضمن لكافة الحالات والمناقلات الممكنة والذي يخضع لعدد من عمليات المراقبة والفلترة المناسبة لتحديد مدى صحة القيود الزمنية. إن حصولنا على مخطط التنفيذ الشامل لا يُتيح لنا مراقبة القيد الزمني مباشرة لأن هذا المخطط غالباً يحتوي عدداً هائلاً من المناقلات الضرورية لعمل النظام والإشارات الإجبارية المكمّلة له ولكنها غير مؤثرة في صحة القيد الزمني المدروس، لذلك بعد حصولنا على هذا المخطط فإننا بحاجة إلى اختزاله بحيث نحافظ فقط على الإشارات والعمليات المرتبطة بالقيد الزمني المطلوب الحكم على صحته وهذا يفرض علينا تحديد هذه العناصر المؤثرة في ملف لتطبيقها كمعايير فلترة خارجية على المخطط لنحصل بالنتيجة على المسار المقابل لهذه المعايير وفق صيغة أتومات يساعد في التحقق من صحة القيد الزمني المطلوب. لذلك يهدف هذا البحث أيضاً إلى معالجة الأتومات المقابل لمخطط الحالات الناتج وتقديمه على شكل أتومات يكافئ المسار المطلوب وفق السلوكية المراقبة ومعالجته أخيراً بشكل بياني بأسلوب يفيد كثيراً في عملية توضيح وتمييز المسار المقابل للقيد الزمني للتأكد من صحته.

## 3.1 أهمية البحث وأهدافه:

يهدف هذا البحث إلى المساعدة على التحقق من أنظمة الزمن الحقيقي والتأكد من احترام القيود الزمنية بحيث لا يوجد مجال للخطأ عند وضع برمجية ما في مرحلة الاستثمار. غالباً ما يؤدي استثمار البرمجية غير المتأكد من صحتها (حتى ولو كان جزئياً) إلى خسائر فادحة بالموارد البشرية أو المادية التي تُمثّل الوسط الأساسي والفعلي لمثل هذه البرمجيات كما أن حدوث أي خطأ في المعالجة قد ينجم عنه إنهاء كامل النظام، فعلى سبيل المثال في نظام مراقبة نبضات قلب مريض وهو نظام زمن حقيقي قد يؤدي أي تباطؤ في عدد النبضات دون حدوث الاستجابة المناسبة وبشكل سريع من البرنامج المختص بهذه المراقبة إلى وفاة ذلك المريض، لذا كانت كتابة أي برنامج ضمن نظام VxWorks ووضعه في مجال التطبيق المباشر دون التحقق منه خطوة خطيرة تُخشى عواقبها.

## 4.1 خطوات البحث:

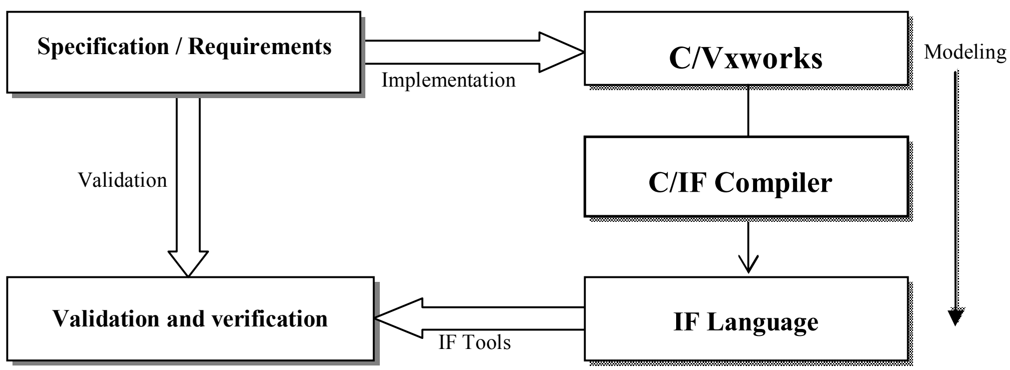
يمر هذا البحث بعدة مراحل أولها هي دراسة لغتي C/VxWorks و IF إضافة إلى التقنيات المستخدمة في بناء المترجمات، ولبناء المترجم المنشود فإننا سنتبع الخطوات التالية (الشكل ١-١) :

أولاً : من أجل بناء المحللين اللفظي والتركيبي سنقوم بكتابة القواعد الخاصة بلغة C التي تُستخدم في بناء أنظمة الزمن الحقيقي المستندة على نظام التشغيل VxWorks.

ثانياً : لتوليد البرنامج الهدف سنقوم بدراسة التقابل بين لغتي الدخل (C/VxWorks) والخرج (IF) بشكل كامل معتمدين في ذلك على المقابلة بين العناصر المشار إليها في بحثنا والتي تم التطرق إليها بشكل جزئي في البحث المقدم في [1].

ثالثاً : بناء المترجم اللازم لعملية التحويل.

رابعاً : اختبار المترجم من خلال ترجمة مجموعة من التطبيقات المكتوبة بلغة دخل المترجم (C/VxWorks)، إلى لغة خرج المترجم (IF). ومن ثم الاستفادة من أدوات لغة IF مثل المحاكي (simulator) الذي يولّد كافة المسارات الممكنة (التي يجب أن تكون مطابقة لسلوك تطبيق الدخل) ومن ثم التأكد من صحة هذه المسارات وذلك بمقارنتها مع نتائج تنفيذ تطبيق الدخل التي حصلنا عليها يدوياً (أي بعد تمرير تطبيق الدخل).



**الشكل(1-1): مخطط عام للبحث**

## 5.1 منهجية البحث:

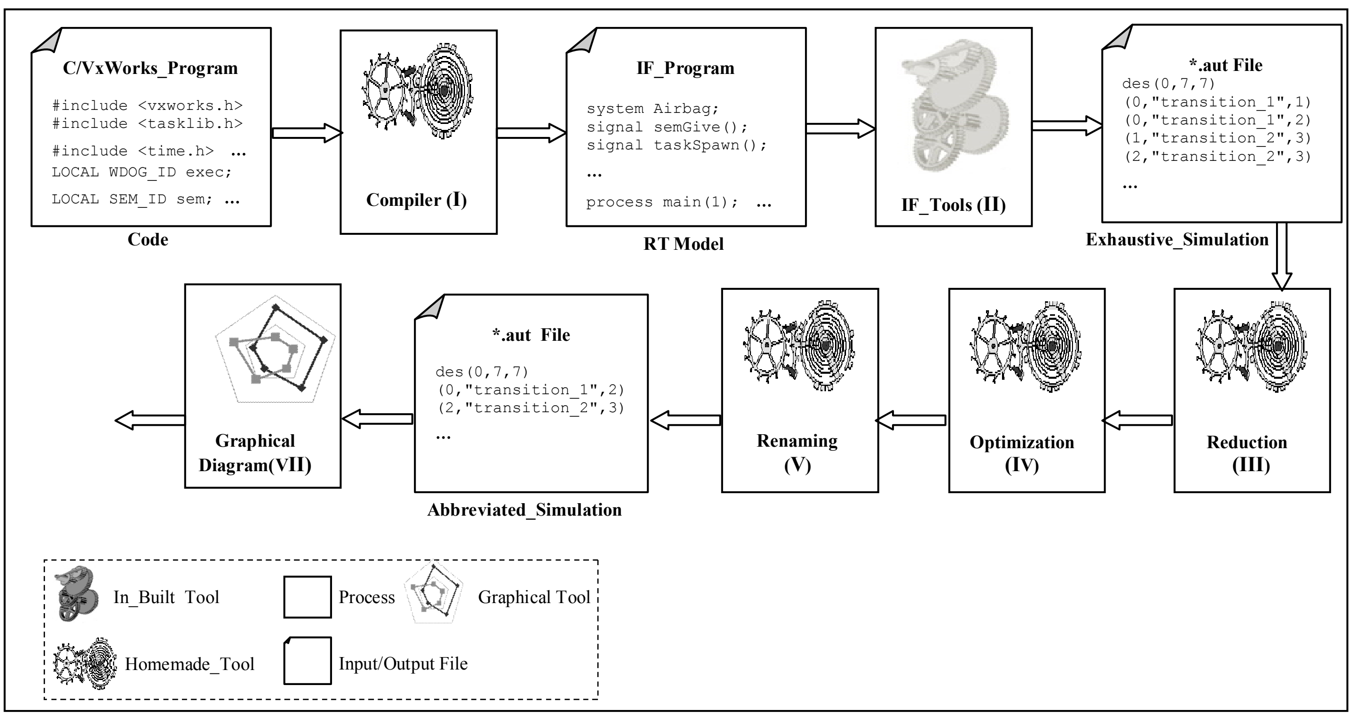
نظراً لدقة أنظمة الزمن الحقيقي وما تفرضه من قيود زمنية خاصة فإنه لابدّ من التحقق من صحة هذه الأنظمة في مرحلة تحقيق الشيفرة البرمجية وهذا هو كما أسلفنا مجال بحثنا الذي يركز على التحقق من صحة أنظمة الزمن الحقيقي قبل استثمارها. فيمايلي نسرد الطريقة المتبعة التي تُمكننا من التحقق من صحة أنظمة الزمن الحقيقي على مستوى الشيفرة.

كما يظهر في (الشكل 1-2) التالي فإنه وانطلاقاً من ملف الدخل المكتوب بلغة (C/VxWorks_Program) ستقوم مرحلة Compiler(I) بتحويل ملف الدخل إلى ملف خرج (IF_Program) وذلك من خلال مترجم خاص سنقوم بعرض مراحل بنائه ضمن سياق بحثنا في الفصول القادمة. في المرحلة التالية IF_Tools(II) وانطلاقاً من ملف الخرج السابق سنقوم باستخدام الأدوات المرافقة للغة IF (التي سنذكرها لاحقاً) من أجل توليد كافة المسارات الممكنة والمكافئة لبرنامج الخرج الناتج عن

مترجمنا وبالتالي نحصل على ملف من نوع AUT يحوي على هذه المسارات (كما هو موضح في الشكل 1-2).

من الملاحظ وخلال دراستنا أن الأتومات الناتج عن الفقرة السابقة يكون بأحجام عملاقة من مرتبة (٤٠٠,٠٠٠ حالة state وأكثر من ١,٠٠٠,٠٠٠ مناقلة transition) وبالتالي فإنه يكون عادة غير مقروء و لا يمكن التحقق من صحة القيوالمعُبّر عنها في ملف الدخل بشكل مباشر من خلالهلذلك كان لا بدّ لنا من معالجة هذا الأتومات وفق عدد من المراحل هي:

- عملية المراقبة (Observation(III التي يتم من خلالها مراقبة مسارات محددة من الأتوماتواختزال المسارات غير المطلوبة.
- عملية التحسين (Optimization(IV التي يتم من خلالها عملية إزالة الغموض من الأتومات الناتج سابقاً بحذف المناقلات المكررة أوالتي تؤدي إلى حالة عدم التحديد.
- عملية إعادة التسمية (Renaming(V التي تهدف إلى إعادة تسمية المناقلات بأسماء قابلة للقراءة.




**الشكل(1-2): طريقة العمل المتبعة في البحث**

وبالتالي نتج لدينا أتومات قابل للقراءة يمكننا من خلاله استخلاص المعلومات المطلوبة بشكل أسهل ومباشر.

- في المرحلة الأخيرة (VII) سنقوم برسم المخطط النهائي المطلوب المقابل للأتومات الناتج بشكل بياني بهدف تحليل النتائج التي حصلنا عليها.

سنستعرض لاحقاً هذه المراحل بشكل مفصّل بحيث سنحدّد الكيفية التي تم من خلالها بناء المترجم مع شرح كافة عمليات المعالجة الأخرى التي تُطبّق على الملف الناتج (خرج مترجمنا) للحصول على المخطط النهائي المعبّر عن القيد الزمني المدروس بشكل واضح ودقيق.

## 6.1 الخطوط العامة للأطروحة:

تقع الأطروحة في سبعة فصول رئيسية تمت صياغتها وفق تسلسل منهجي علمي يساعد القارئ على سهولة تتبع خطوات البحث وأفكاره، وتعرض بأسلوب واضح ومرن المراحل المختلفة التي تطوّر وفقها البحث والأدوات البرمجية المعتمدة لإنجازه مع تطبيق عدة اختبارات عملية شاملة للتأكد من صحة نتائجه النهائية وتحقيقه للهدف المطلوب منه.

بعد إتمام الفصل الأول الذي يتضمن مقدمة عامة عن أهمية البحث ومجال تطبيقه، يتحدث الفصل الثاني عن أنظمة الزمن الحقيقي ومواصفاتها والشروط الزمنية التي تُقيّدها والبيئات المختلفة المرتبطة بها والتي تم اعتمادها لإنجاز بحثنا هذا، مع شرح عام للأدوات واللغات البرمجية التي تساعد في التحقق من صحة هذه البرامج خلال مرحلة تحقيق الشيفرة النهائية. كما تم وفقه مناقشة ضرورة بناء مترجم وسيط كخطوة داعمة يلبي الحاجة البرمجية المطروحة في التحويل بين تعليمات اللغتين المعتمدتين في البحث.

يتحدث الفصل الثالث عن مراحل بناء المترجم والأدوات المستخدمة في ذلك مع توضيح الفائدة المرجوة منه بما يخدم بحثنا، كما يتضمن عملية تعريف بقواعد لغة C/VxWorks مصاغة بطريقة الـ BNF(Backus-Naur Form) وكيف تم استخدام هذه القواعد في مرحلتي التحليل اللفظي والتركيبي مع شرح تفصيلي للخطوات العملية المنجزة للحصول على محلل لفظي وتركيبي شامل يُنفذ عمله بدقة وبشكل أمثلي ويعطي نتائج صحيحة.

ويتحدث الفصل الرابع عن مرحلة توليد الشيفرة البرمجية النهائية وكيفية تحديد التعليمات المتقابلة بين دخل مترجمنا (لغة C/VxWorks) وخرجه (لغة IF)، كما تم وفقه إيضاح الأسلوب العملي المتبع لإجراء المقابلة المطلوبة وكيف تم توليد الشيفرة النهائية لمترجمنا وتوصيف البنى البرمجية الخاصة به والتي تفيد في الحصول على الملف التنفيذي النهائي، إضافة إلى القيام بعملية الاختبار الفعلي لنتائج مترجمنا والتأكد من صحة عمله.

يتحدث الفصل الخامس عن كيفية معالجة الأتومات الناتج وفق خوارزميات خاصة تساعد في إيضاح المسار المراد التحقق من صحة قيوده الزمنية بشكل مباشر.

يتناول الفصل السادس عدداً من الحالات الدراسية التي تؤكد صحة نتائج بحثنا من خلال اختبار تفصيلي لكافة مراحله وذلك بتنفيذ تطبيق عملي شامل لنظام الحقيبة الهوائية Airbag تم وفقه تتبع مسارات التنفيذ المختلفة والموافقة للسيناريوهات المدروسة ذات القيود الزمنية المُراقبة إضافة إلى اختبار ذلك وفق تطبيق آخر يتناول عمل قارئ السيدي (CD Reader) من وجهة نظر الزمن الحقيقي.

يتحدث الفصل السابع عن الملخص العام لهذه الأطروحة وآفاق التطوير المستقبلية المطروحة.

أخيراً، نجد الملاحق الأربعة، حيث يتضمن الملحق I لائحة بقواعد لغة C/VxWorks مصاغة وفق طريقة BNF. أما الملحق II فيتضمن مقابلة بين أهم توابع لغة C/VxWorks ولغة IF. ويتضمن الملحق III شرحاً لواجهات المترجم مع واجهات التطبيق العملي الناتج عن هذا البحث.

يتضمن الملحق الأخير IV جزءاً من الشيفرة البرمجية الخاصة بخوارزميات المعالجة المقترحة للتحقق من صحة القيود الزمنية.

## 7.1 الخلاصة:

بعد هذه المقدمة التي تحدثنا فيها عن الخطوط العامة للبحث والأهداف المرجوة منه، التي تتلخص بعملية بناء المترجم الذي يقوم بتحويل التعليمات بين C/VxWorks و IF ومن ثم معالجة مخطط المحاكاة الشامل للتنفيذ وفق أدوات برمجية مختلفة تساعد في التحقق من صحة القيود الزمنية المفروضة على النظام، سنقوم الآن بالتعرف على الأدوات البرمجية المختلفة المعتمدة خلال بحثنا وما تتميز به كل بيئة أو لغة برمجية تم استخدامها مع شرح الهدف العام لاستخدامها ضمن بحثنا وأسلوب عمل كل منها.

# ٢

# الفصل الثاني

# أنظمة الزمن الحقيقي وأدواته

# **R**eal **T**ime **S**ystems and **T**ools

1.2 مقدمة.

2.2 تعريف أنظمة الزمن الحقيقي.

3.2 بناء نظام زمن حقيقي.

4.2 نظام التشغيل VXWORKS.

5.2 عناصر الزمن الحقيقي وبرمجتها في نظام VXWORKS.

6.2 مراحل تطوير نظام الزمن الحقيقي.

7.2 لغة IF (INTERMEDIATE FORMULA LANGUAGE).

8.2 أدوات برمجية داعمة.

9.2 الخاتمة.

# أنظمة الزمن الحقيقي وأدواته

## 1.2 مقدمة:

شهدت العقود الماضية العديد من الأبحاث والدراسات التي اهتمت بمفهوم الجدولة الزمنية [2] للمهام التفرعية وكيفية تحديد أولويات تنفيذها وأسلوب معالجة التداخلات الزمنية بينها، بدأت عام 1969 بمحاولة الباحث Hoare طرح تقنيات خاصة تدعم توصيف الصيغ الزمنية، وفي عام 1973 تم من قبل Liu&Layland مناقشة مفهوم تحليل وجدولة نظم الزمن الحقيقي بشكل أعطى تبرمرجعاً أساسياً للدراسات اللاحقة. بقيت صحة أي برمجية تُقاس بإمكانية وصولها لمرحلة التنفيذ فقط إلى أن أشار Wirth عام 1977 إلى ضرورة التمييز بين صحة البرمجية قواعدياً ومفهوم تحقيقها للقيود الزمنية المفروضة عليها يُقال إنها برمجية تُراعي الزمن الحقيقي، فيما بعد في عام 1983 وضّح Mok الصعوبات التي تعترض تطبيق مفاهيم الزمن الحقيقي ضمن الأنظمة البرمجية وكيف أنها تحتاج متطلبات مادية ذات وثوقية عالية إضافة إلى أنه طرح فكرة تطبيق البرمجة متعددة المعالجات، وبذلك بدأت الخصائص الزمنية للنظام تؤخذ برمجياً بعين الاعتبار ففي عام 1988 حدد Turski الزمن بأنه القضية الأهم التي ينبغي احترامها خلال المراحل المختلفة لدورة حياة أي برمجية، وفي عام 1989 وضّح Hehner أن القيود الزمنية المفروضة على أي نظام يجب أن تُراعى وتُعتمد في تحديد البنى البرمجية المكوّنة له منذ اللحظة الأولى لبنائه لأن ذلك يعطي نتائج أدق مما لو تم كتابة برنامج عام ومن ثم تُختبر فيه هذه القيود، ثم ظهرت نظريات التزامن عام 1992 من قبل Berry&Gonthier وطُبّقت ضمن الأنظمة وحيدة المعالج وافترضت الاستجابة الفورية للأحداث الخارجية التي قد تعترض تنفيذ أي نظام، كما حُددت مفاهيم الأنظمة اللامتزامنة التي تتم استجابتها ضمن فترات زمنية متقاربة وطُبّقت ضمن الأنظمة متعددة المعالجات، والأنظمة الموزّعة. عام 1992 عرّف Douglas Jensen أي نظام حاسوبي بأنه نظام زمن حقيقي يتطلب إدارة خاصة للموارد بما يتوافق مع القيود الزمنية المفروضة عليه تطوّرت هذه المفاهيم فيما بعد كثيراً إلى أن تبلورت وفق صورتها الحالية وما تزال عجلة العلم مستمرة دون توقف بشكل نعجز معه عن رسم حدود النهاية.

## 2.2 تعريف أنظمة الزمن الحقيقي:

تُعرَّف أنظمة الزمن الحقيقي بأنها الأنظمة التي لا تتوقف صحتها على صحة نتائج حساباتها فقط وإنما على الزمن الذي أُنجزت به هذه النتائج [3]. من ناحية أخرى يمكن أن تتصف هذه الأنظمة بثلاث مزايا رئيسة هي:

- التنبؤية Predictability: يمكننا التنبؤ بأسلوب عملها مما يتيح لنا تحكماً أشمل وتحديداً أدق لزمن تنفيذها إذ أنها تحتوي على نقاط حرجة يجب الاهتمام بها للتنفيذ الصحيح.
- الحتمية أو المحدودية Deterministic: حيث يمكننا ضمنها معرفة الزمن اللازم لتنفيذ عمل ما أي أنها أنظمة خالية من الغموض Ambiguity.
- الوثوقية Reliability: وهي من أهم الصفات التي تتمتع بها أنظمة الزمن الحقيقي.

كما أن أنظمة الزمن الحقيقي غالباً ماتكون أنظمة مدمجة Embedded System مضمنة ضمن بيئة عمل معينة وتقوم بعمل خاص ومحدود.

من التعاريف الأخرى الشائعة لأنظمة الزمن الحقيقي أنها أنظمة حاسوبية (computer system) تأخذ بعين الاعتبار القيود الزمنية وتُصنّف وفق العناصر الأساسية المكونة لها (الأفعال (actions) والأحداث (Events) والمعطيات (Data))[4] إلى عدة تصنيفات فمثلاً نجد أن للمعطيات عمراً (Data Age) يمثل الفترة الزمنية التي تبقى ضمنها هذه المعطيات صالحة للاستخدام كما أن هناك زمناً أقصى للاستجابة المتاحة للأحداث الخارجية تُحدّد بقيمة الـ Deadline الذي يُعتبر أهم قيد زمني وبناءً على هذا القيد و القيود الزمنية الأخرى يمكن أن تُصنّف أنظمة الزمن الحقيقي إلى:

- أنظمة الزمن الحقيقي القاسية (Hard Real time system): يجب أن تتم استجابتها للأحداث الخارجية التي تعترضها خلال فترة زمنية معينة بالقيمة Deadline ولا تُقبل أي استجابة بعد ذلك الزمن،تُعتبر هذه الأنظمة مُقادة بالزمن على عكس الأنظمة المقادة بالأحداث (التي تتوقف استجابتها على حالة النظام المحددة قبل تفعيل الحدث) والأنظمة التفاعلية (reactive) [5] التي تتحدد ضمنها الاستجابة لأي حدث بزمن خاص به، كأنظمة التحكم.
- أنظمة الزمن الحقيقي المرنة (Soft Real time system): وهي التي تستمر في عملها رغم تجاوز الفترة الزمنية المتاحة للاستجابة الممكنة.

أما من ناحية الأبعاد الزمنية المتاحة لتنفيذ القيود التي قد تُفرض على هذه الانظمة فقد تم تصنيفها إلى:

- الأنظمة واسعة التطبيق (Broad system): يمكن تحقيق متطلباتها الزمنية بسهولة دون الحاجة إلى تقنيات خاصة (القيد الزمني واسع المجال).

- الأنظمة ضيقة التطبيق (Narrow system): تكون قيودها الزمنية ضيقة (متلاصقة) وتحتاج إلى دقة في التصميم والتنفيذ وعناية خاصة لتأمين متطلباتها وتحقيق قيودها علماً أن أي تأخير في تحقيق تلك القيود يمكن أن يؤدي إلى فشل كامل للنظام.

## 3.2 بناء نظام زمن حقيقي:

يحتاج بناء هذا النظام مجموعة من المتطلبات البرمجية والعتاد الصلب(Hardware and Software) لذا كان لابدّ من توفر عتاد صلب موثوق وذي جودة عالية وكافية لتحقيق الأداء المطلوب، فعلى سبيل المثال: نحتاج تأمين نوع محدد من شبكات CAN ذات مواصفات خاصة (مكونة من ثلاث طبقات، متعددة النقاط، عنوان وحيد للرسالة)، وأجهزة طرفية متكاملة (المتحكمات بالدخل والخرج I/O controller،محوّلات A/D converter، ومؤقت زمني حقيقي clock، وحدات تخزينية لحفظ المعلومات ذواكر (Rom, EEPROM, FLASH, RAM) ومعالج بسرعة معينة وتردد خاص (8/16 bits). إضافة إلى ضرورة اعتماد لغة برمجة تدعم نظم الزمن الحقيقي تكون قريبة من لغة الآلة، إذ غالباً ماتُعتمد لغة C أو لغة المجمع Assembly في بناء هذه التطبيقات وذلك لما تمتلكه من كفاءة برمجية في تعليماتها التيتُراعي سهولة إدارة الذاكرة والوصول لكافة مداخل النظام ومخارجه. كما أنهاتُعد لغة واسعة الانتشار فهي تتيح التحكم بالعمليات (processes) التي تتم ضمن أنظمة التشغيل، ومن أنظمة التشغيل التي تدعم هذه التطبيقات:

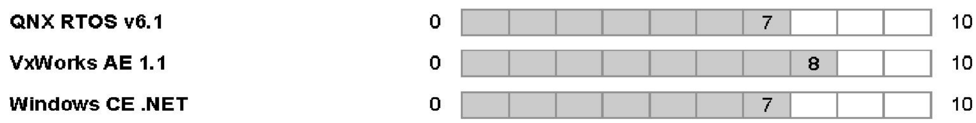
- VxWorks from Wind River Systems.
- LynxOS from Lynx Real Time Systems.
- QNX from QNX software systems.
- PSOS fom integrated systems.
- RTC from GSI TECSI.
- iRmx from intel .
- Chorus from chorus (France).

أكثر نظم الزمن الحقيقي شهرة هما نظاما QNX وVxWorks ولقد قمنا في بحثنا هذا بالاعتماد على نظام التشغيل VxWorks [6] للتعامل مع أنظمة الزمن الحقيقي.

## 4.2 نظام التشغيل VxWorks:

هو نظام تشغيل زمن حقيقي [7] عالي الأداء مخصص للتعامل مع الأنظمة المضمنة وأنظمة المعالجاتالمصغّرة، تمّتطوير ه من قبل شركة Wind River في كاليفورنيا ويتصف بعدد من المزايا فهو يعمل على أهم المنصات التيتُستعمل في الأنظمة المضمنة مثل x86 وMIPS وPowerPC وSH-4 وARMويُعتبر بمثابة نواة أساسية لمعالجة التطبيقات متعددة المهام وجدولتها زمنياً أو وفق أولوية التنفيذ إضافة إلى إمكانية التعامل مع المعالجات المتعددة المتزامنة SMP بما يؤمنه من استجابة سريعة للمقاطعات

وحماية كبيرة للذاكرة (يوضح الشكل 2-1 مقارنة لمصادر API) ويتوافق نظام VxWorks مع واجهات أنظمة التشغيل القابلة للنقل IEEE's POSIX(Portable Operating System Interface) (يوضح الجدول 2-1 مقارنة لمواصفات نظام VxWorks) [8]، ومن خلال دعمه للغة ANSI C ولغة C++ المساعدة في معالجة الاستثناءات (exception) يتيح مرونة فيتواصل العمليات المتزامنة في وقت التنفيذ وسرعة في معالجة الأخطاء إضافة إلى أنه يحوي محاكي VxSim ويدعم شبكة IPv6.

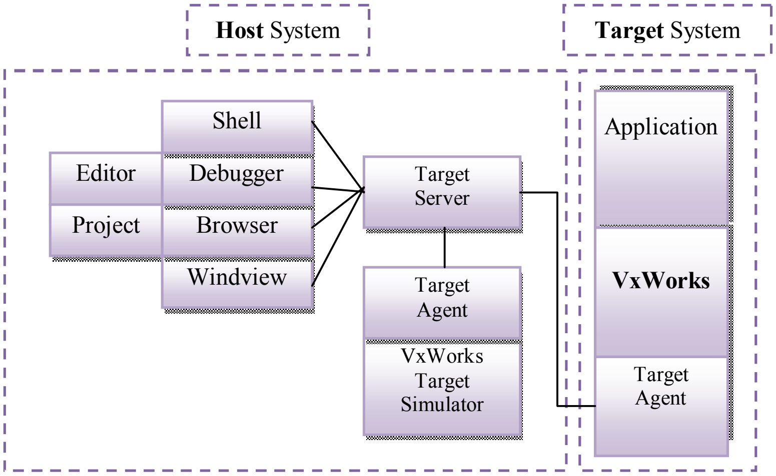



الشكل(2-1): مقارنة مصادر API

| | QNX RTOS v6.1 | VxWorks AE 1.1 | Windows CE .NET |
|---|---|---|---|
| **MMU support** | Yes | Yes | Yes |
| **Physical page size** | Depends on architecture | Depends on architecture | Depends on architecture |
| **Swapping/Demand Paging** | Yes/No | Never | Supported, but can be disabled to achieve real-time performance |
| **Virtual memory** | Yes | Yes | Yes |
| **Memory protection models** | Full virtual Memory protection | Full virtual Memory protection | Full virtual Memory protection |
| **Task management** | 82% | 88% | 71% |
| **Timer** | 100% | 100% | 67% |
| **Memory management** | 27% | 77% | 35% |
| **Semaphore** | 35% | 90% | 40% |
| **Mutex** | 67% | 92% | 67% |
| **Message Queue and Mailbox** | 41% | 81% | 75% |
| **POSIX signals** | 100% | 78% | 0% |

الجدول(2-1): مقارنة VxWorks مع نظم تشغيل أخرى

يُعتبر VxWorks أحد العناصر الأساسية في بيئة Tornado [9] التي هي عبارة عن أداة برمجية متكاملة لتطوير البرمجيات تدعم بشكل كبير الأنظمة المضمنة (embedded systems) ونظم الزمن الحقيقي وتسهّل عملية الاتصال بين بنى الشبكات المختلفة وتُتيح البرمجة بلغة C ضمن نظام التشغيل VxWorks من خلال تأمينها للخدّم الهدف (target server) الذي يتولى عملية الاتصال ويتضمن حكماً نظام VxWorks إضافة إلى تأمين منصة عمل مضيفة (host) خاصة بتنفيذ التطبيقات البرمجية قد تتضمن هذه المنصة أي نظام تشغيل آخر مثل windows أو Linux عدا نظام VxWorks، وتتوضح العلاقات [10] بين الأدوات التفاعلية للنظام المضيف والنظام الهدف وفق الشكل التالي:




الشكل(٢-2): بيئة Tornado متضمنة نظام التشغيل VxWorks

## 5.2 عناصر الزمن الحقيقي وبرمجتها في نظام VxWorks:

تقسم العناصر المكونة لأنظمة الزمن الحقيقي [11] إلى مجموعتين: العناصر القابلة للبرمجة والعناصر القابلة للتهيئة. تُمثّل العناصر القابلة للبرمجة الأجزاء الرئيسية التي يتكون منها البرنامج وهي:

- المهمة (task): عبارة عن كتلة برمجية تمر بمجموعة حالات (سنتطرق إليها لاحقاً) وتتألف من مجموعة تعليمات تُنفّذ بشكل تسلسلي لأداء عملها وتحجز حيزاً معيناً من الذاكرة، لكل مهمة أولويتها الخاصة التي تُحدّد ترتيب تنفيذها.
- برنامج خدمة المقاطعة (ISR (Interrupt Service Routine: وهو مجموعة من التعليمات التسلسلية تُنجز مهمة محددة ضمن النظام وتُنفّذ عند ورود حدث (مقاطعة) من الوسط الخارجي.
- برنامج خدمة المؤقت (ASR (Alarm Service Routine: وهو روتين يرتبط تنفيذه بالزمن (يُطبق عليه نفس القيود التي تطبق على برنامج خدمة المقاطعة).

أما القسم الثاني وهو العناصر القابلة للتهيئة فهي تؤمن التزامن والاتصال بين العناصر القابلة للبرمجة آنفة الذكر وهذه العناصر هي:

- السيمافور(Semaphore): هو وسيلة اتصال وتزامن، يأخذ قيمة محددة تدل على عدد المهام التي يُسمح لها بالدخول إلى المقطع الحرج (critical section) في كل لحظة، لكل سيمافور خط انتظار خاص به تنتظم فيه المهام التي يتوقف تنفيذها على تحرير المصدر المرافق. للسيمافور ثلاثة أنواع وهي الثنائية (Binary Semaphore) التي تخزن قيمة وحيدة منطقية تدل فيما إذا كان المقطع الحرج متاحاً أم مشغولاً، والتعدادية (Counting Semaphore) التي تمتلك رتلاً تخزن فيه الإشارات لتعالج فيما بعد كما يحتفظ بعدد مرات تحرير المصدر،وأخيراً ذات الاستبعاد المتبادل (Mutex or

(Mutual exclusion Semaphore يعمل بشكل مشابه للثنائية غير أنه مخصص لحماية مصادر النظام.

- أما العنصر الثاني فهو علبة البريد (MailBox): وتمتاز بتخزينها للمعلومات إضافة للإشارات بطريقة الرتل الذكي، مما يؤمن وسيلةَ اتصال (تبادل معلومات) وتزامناً بين المهام (تدخل المهمة الطالبة للرسالة وضعية انتظار ريثما يتم استقبالها الرسالة المطلوبة).
- العنصر الثالث هو الحدث (Event): الذي يمكن أن يُمثّل ببوابة AND أو OR متعددة المداخل أي يأخذ قيمة منطقية تُعبّر عن صحة أو خطأ حالة ما.
- أخيراً المصدر (Resource): يستخدم من أجل تمثيل المصادر المتشاركة مثل المتحولات وبنى المعطيات أو الملفات أو الأجهزة الخارجية مثل شاشة العرض LCD، يتم حماية المصدر باستخدام السيمافور أو حجب المقاطعات أو بتطبيق مفهوم المنطقة المحمية (التي يتم وفقها تغليف مجموعة مهام لا يمكن مقاطعتها حتى تمام تنفيذها).

تتضمن بيئة Tornado عدداً من المكتبات البرمجية التي تحتوي توابع جاهزة للاستخدام المباشر من قبل المبرمج، مما يُتيح سهولة برمجة أداء مكونات الزمن الحقيقي وفق C/VxWorks بما يتناسب مع الهدف البرمجي المطلوب للتطبيق المنفّذ. سنعرض حالياً أهم المكتبات مع روتيناتها البرمجية التي تم أخذها بعين الاعتبار في بناء مترجمنا وشرح بسيط لوظيفة كل منها، الجدول(٢-٢):

| المكتبة | التابع | الوظيفة البرمجية التي يؤديها |
|---|---|---|
| المهمة taskLib | taskSpawn | إنشاء وتفعيل المهمة بأفضلية وخيارات محددة مع مؤشر إلى سجل تخزينها FUNCPTR. |
| | taskInit | تهيئة منطقة محددة من الذاكرة من أجل حيز خاص تحجزه المهمة لاحقاً خلال عملها . |
| | taskActivate | تفعيل مهمة بعد تهيئتها وفق الإجراء السابق. |
| | taskDelay | تأخير المهمة المنفذة لنفسها فترة من الزمن (وضعية إيقاف مؤقت) حتى انتهاء زمن التأخير. |
| | taskResume | إعادة المهمة المعلقة إلى حالة العمل لمتابعة أعمالها. |
| | taskDelete | حذف المهمة ذات المعرّف المرافق لهذا التابع. |
| | taskSuspend | تعليق مهمة ما. |
| المؤقت wdLib | wdCreate | إنشاء المؤقت مع إعادة قيمة معرّف خاص به wdId. |
| | wdStart | تفعيل المؤقت ذي المعرّف wdId مع تحديد اسم تابع خدمة المؤقت pRoutine نمط FUNCPTR. |
| | wdDelete | حذف المؤقت ذي المعرّف wdId المرافق لهذا التابع. |
| | wdCancel | إيقاف المؤقت دون إلغائه أو حذفه إذ يمكن إعادة تفعيله ثانية. |

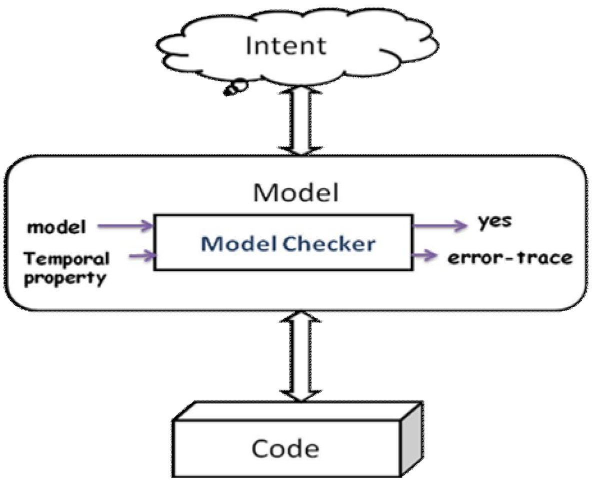
| المكتبة | التابع | الوظيفة البرمجية التي يؤديها |
|---|---|---|
| السيمافور semBLib | semTake | حجز السيمافور من قبل مهمة ما. |
| | semGive | تحرير السيمافور المحجوز من قبل المهمة الحالية وجعله متاحاً للمهام الأخرى. |
| | semDelete | حذف السيمافور ذي المعرّف المحدد بهذا التابع. |
| | semBCreate | إنشاء سيمافور ثنائي بحالة ابتدائية full أو empty ويتحدد ذلك وفق قيمة المتحول المرافق. |
| semCLib | semCCreate | إنشاء سيمافور تعدادي يترافق بقيمة ابتدائية لعدّاد خاص initial count. |
| semMLib | semMCreate | إنشاء سيمافور المنع التبادلي بالحالة الابتدائية full. |
| علبة البريد msgQLib | msgQCreate | إنشاء علبة البريد وجعلها جاهزة لاستقبال الرسائل. |
| | msgQSend | إرسال رسالة إلى علبة البريد (يترافق بمتحول يدل على زمن التأخير المسموح للاستجابة المنتظرة). |
| | msgQReceive | القراءة من علبة البريد (يترافق بمتحول يدل على زمن التأخير المسموح للاستجابة المنتظرة). |
| | msgQDelete | حذف رتل الرسائل ذي المعرّف msgQId الموافق لهذا التابع. |
| برنامج خدمة المقاطعة intArchLib | intConnect | إنشاء مقاطعة مع ربط شعاع المقاطعة ببرنامج خدمة المقاطعة ItRoutine. |
| | intEnable | إعادة تفعيل المقاطعة التي تم تعطيلها سابقاً. |
| | intDisable | تعطيل كافة المقاطعات قيد العمل حتى انتهاء المهمة الحالية. |
| | intLock | تطبيق عملية قفل على برنامج خدمة المقاطعة (يعيد مدخلاً للمقاطعة المستقبلة للقفل). |
| | intUnlock | فتح القفل عن المقاطعة يتم من قبل مهمة أخرى حصراً (لا يطبق داخل برنامج خدمة المقاطعة). |

**الجدول(٢-2): أهم مكتبات C/VxWorks والتوابع الموافقة لها**

## 6.2 مراحل تطوير نظام الزمن الحقيقي:

تمر أي برمجية خلال دورة حياتها بمجموعة من مراحل التطوير: تبدأ بالتحليل ومن ثم التصميم ثم كتابة الشيفرة البرمجية واختبارها لتقديمها للاستثمار النهائي [١٢]. يتم في مرحلة التحليل توصيف الوظائف الواجب على النظام القيام بها إضافة إلى القيود الزمنية التي يجب مراعاتها خلال عمل النظام مثل Deadlocks,Timelocks والتيتُعد إحدى المشاكل الأساسية التي تعترض أنظمة البرمجة المتداخلة المتعددة المهام، ويمكن في هذه المرحلة استخدام لغة النمذجة الموحدة UML (Unified Modeling Language) وهي لغة قياسية لتوصيف ونمذجة المعطيات تستخدم مفاهيم البرمجة غرضية التوجه وقد جاءت لتوحد لغات النمذجة المختلفة بما تقدمه من مخططات تحليلية مبنية وفق أدوات ذات دلالات توصيفية خاصة. أما في مرحلة التصميم فيتم توضيح مجموعة من التفاصيل وتخصيص القيود الزمنية بشكل أكبر دون الدخول بالتحقيق النهائي للشيفرة البرمجية ويمكننا في هذه المرحلة اعتماد لغة التوصيف الشكلية SDL (Specification and Description Language) [13] التيتُستخدم في الأنظمةالموز عة من أجل

العمليات المتزامنة وغير المتزامنة. هناك العديد من التقنيات المعتمدة في التحقق من صحة مرحلتي التحليل والتصميموتُعد تقنية فحص الموديل(model checking) [14] الأكثر استخداماً في هذا المجال الشكل(٢-3)تُطبّق للتحقق من أنظمة التحكم بالزمن الحقيقي والدارات الزمنية اللامتزامنة والخوارزميات الموزعة ومفاهيم الجدولة الزمنية حيثتُقدم استعراضاً كاملاً لحالات النظامعتمدةً على مفاهيم الأتومات الزمني الذي يدرس الحالات الممكنة لعناصر النموذج (التقاطعات العملية للمهام المطلوبة من النظام مع تداخلاتها المفروضة والمحتملة) ومدى مطابقتها للقيود الزمنية التي تحكم النظام مع تحديد مسارات الانتقال غير الصحيحة.

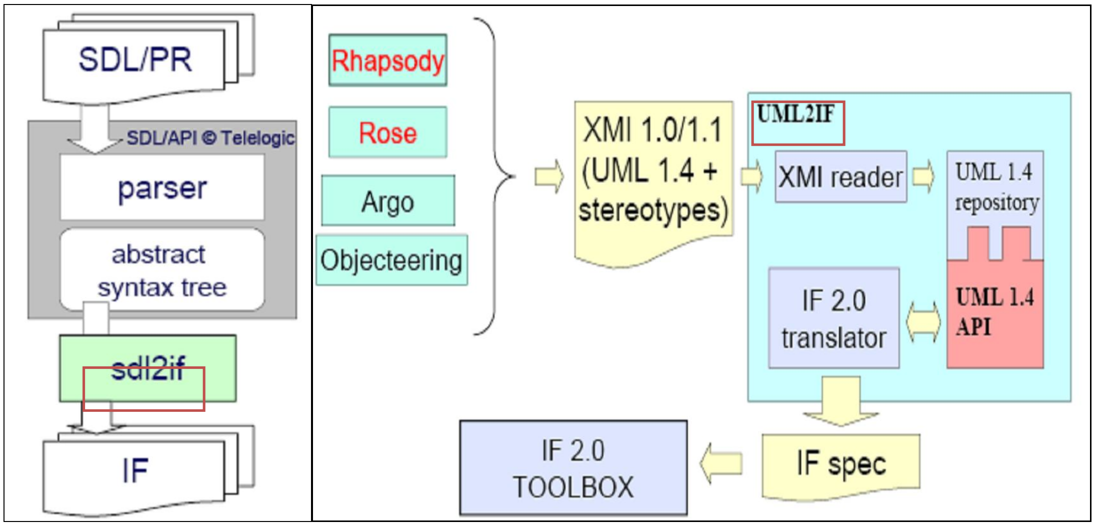



**الشكل(٢-٣): فاحص الموديل (model checker)**

بعد الانتهاء من مرحلة التصميم تأتي مرحلة التحقيق وهي المرحلة الأكثر أهمية في حياة البرمجية. المشكلة التي تواجهنا في أنظمة الزمن الحقيقي هي الافتقار للأدوات التي تقوم بشكل آلي بنقل التحليل (الذي تم التأكد من صحته باستخدام فاحص الموديل) والتصميم المقابل له (الذي تم التأكد من صحتهأيضاً بالطريقة نفسها) إلى الشيفرة النهائية التي تحققه، وبما أن مرحلة التحقيق النهائي للبرمجية (أي كتابة الشيفرة التنفيذية للبرنامج الموافق لها) هي الأكثر أهمية في عملنا البرمجي لذا كان لابد لنا من التأكد من صحتها بصورة خاصة.وفقاً لذلك قمنا ببناء مترجميحوّل الشيفرة المكتوبة والقابلة للتنفيذ وفق لغة C/VxWorks إلى لغة زمن حقيقي وسيطة مستخدمة بكثرة في مجال التأكد من صحة أنظمة الزمن الحقيقي هي لغة IF. مع وجود مترجمنا هذا أصبح بإمكاننا استخدام لغة IF لإثبات صحة هذه البرمجيات وتفادي أي مخاطر قد تنجم عن عدم مراعاة أحد القيود الزمنية لتعليمات البرنامج والتأكد أن ما أنجزناه من مجهود برمجي يتوافق مع الهدف الفعلي للعمل المطلوب.

## 7.2 لغة IF (Intermediate Formula language):

بدأت المحاولات الأولى لتعريف لغة نمذجة للزمن الحقيقي عام (١٩٨٨) حين حدّد مركز الأبحاث المتقدم verimag تعليمات عامة للتعبير عن الاتصالات اللاتزامنية، وطُوّرت هذه التعليمات وفق صيغة توصيفية محددة، واعتُمدت كلغة برمجة (IF-1.0) خاصة للتعامل مع أدوات التحقق من صحة البرامج، ثم ما لبثت أن تطورت ثانية لتصبح قادرة (IF-2.0) على نمذجة وتحليل الأنظمة التي تعتمد على المعالجة الديناميكية للعمليات البرمجية وبنى المعطيات، وفي عام ٢٠٠١ حصلنا وفق مشروع OMEGA على بيئة محاكاة مرئية متكاملة (IFx-2.0) حيث تم خلاله تحديد تقنيات توصيف شكلية لأنظمة الزمن الحقيقي والأنظمة المضمنة تساعد في التحقق من صحة النماذج الزمنية الموصّفة تحليلياً وفق لغة UML أو المصممة وفق لغة SDL إذ تم تزويد لغة IF وفقه بأدوات وعناصر برمجية داعمة لتحويل وترجمة ملفات SDL ومخططات UML دُعيت بمحرر UML2IF ومحرر SDL2IF (الشكل ٢-٤) [15]. يتضمن محرر UML2IF مترجماً خاصاً بمخططات الـ UML [16] مكتوباً بصيغة XML ويعتمد الأنماط القياسية للغة UML، يعمل هذا المترجم على تحليل وترجمة مخططات UML التي غالباً ما يتم توصيفها باستخدام أدوات برمجية خاصة مثل برنامج Rational Rose أو I-LogixRhapsody ومقابلتها بمكافئٍ لها وفق لغة XML لتتم معالجة هذه الملفات لاحقاً باعتماد (مكتبات UML-API، مفسر IF2.0، محرر نصوص XML). أما مترجم لغة SDL فهو يتضمن مفسراً خاصاً بهذه اللغة يقوم بتحليل شجرة القواعد التجريدية وتفسير معطياتها إلى لغة IF، كما تحتوي لغة IF إضافة لما سبق على بعض أدوات التحليل الستاتيكية وأدوات المحاكاة وفاحص الموديل [17] وواجهات تفاعلية مع المستخدم تُسهّل تطبيق عملية المحاكاة المطلوبة على هذه المخططات ومراقبة أجزاء خاصة منها.

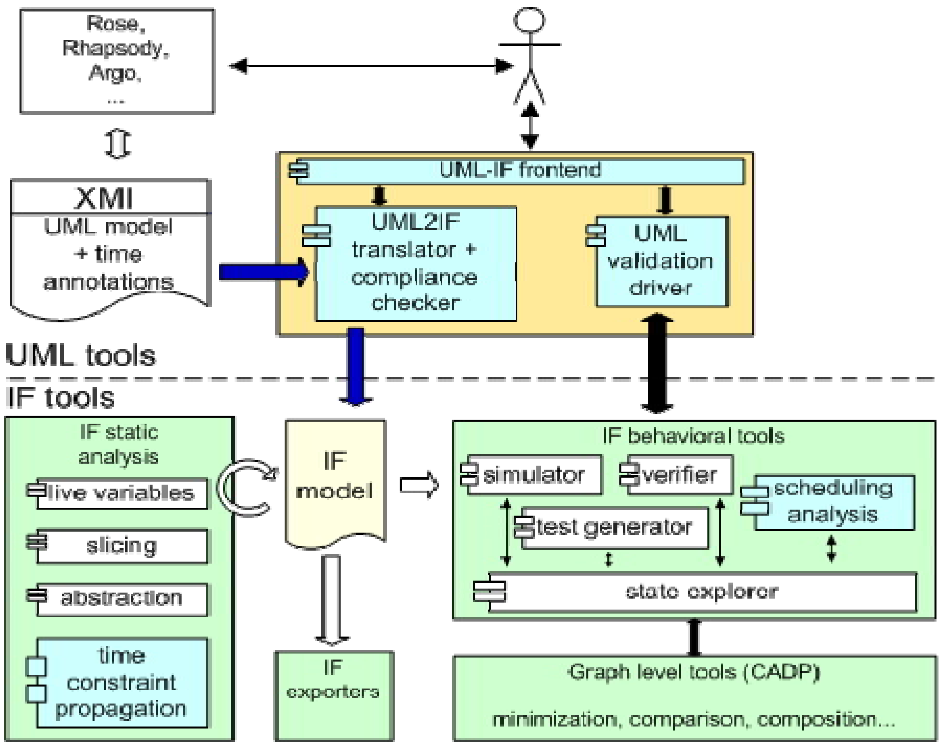



الشكل(٢-٤): مكونات محرر UML2IF و SDL2IF

هكذا تُعتبر لغة IF [18] لغة وسيطة بين لغات التوصيف عالية المستوى مثل SDL أو لغة LOTOS [19] والنماذج الرياضية المعتمدة للاختبار كالأتومات الزمني [20] وأنظمة الانتقال للحالة LTS (Labeled Transition System)، تسمح هذه اللغة بتوصيف مفاهيم الزمن الحقيقي وصيغ معطياته و العمليات المتوازية (من خلال مفهوم السيمافور) وغير المتوازية (تعتمد حواً تخزينية للمعلومات Buffers) أو من خلال التصريح عن متحولات مشتركة بين عمليات النظام تتكوّن من تعليمات برمجية خاصة تعتمد على أساس الأتومات الزمني [21] في التعبير عن قواعد اللغة وفق مبدأ التعابير المنتظمة، كما أنها تُقدّم منصة عمل فريدة تسمح بالتعامل مع تطبيقات البرمجة الوظائفية وغير الوظائفية. يمكن لهذه اللغة أن تُزودنا بالوظائف التالية:

- المحاكاة: تفحص حالات النظام وتوضح مراحل تنفيذه خطوة تلو الأخرى، تحدد نقاط توقف شرطية مع حلول يدوية (بديلة) للحالات غير الحتمية، تعيد جدولة السيناريوهات الممكنة للنظام، تتحكم باستراتيجية الجدولة الزمنية والبارامترات المختلفة المرتبطة بها، وغيرها من الإمكانيات الأخرى.
- التحقق من مرونة القيود الزمنية (deadlocks,timelocks) وصحة تطبيقها في النظام فهي تسمح بتوليد مسارات تشخيصية شاملة للنظام يمكن اختبارها من قبل المحاكي.
- التأكد من صحة توصيف خصائص سلوك النظام (باستخدام فحص الموديل). من خلال:
  - مراقبة الأغراض: تُحدد مخطط الحالات المتفاعلة مع الأحداث والشروط الزمنية مع مراقبة القيود الزمنية الفاصلة بين أحداث النظام.
  - التأكد من صحة صيغ القوانين الرياضية (التكامل والتفاضل) المطبقة ضمن النظام.

يُوضح الشكل (٢-٥) التالي عناصر بيئة IF مثل أدوات تحليل سلوكية النظام ونمذجته (المحاكي، مولّد الاختبار، مستعرض الحالة، المجدول، المتحقق من الصحة)، أدوات التحليل الستاتيكي [22] التي تدرس فترة حياة المتحول ومدى استمرارية القيود الزمنية. إضافة إلى مجموعة النماذج المطورة المقدمة من قبل OMEGA التي تدعم لغة UML والتي تُتيح تفسير مخططات UML وتحليلها وفحصها والتحقق من صحتها ومن ثم ترجمتها إلى لغة IF كما تدعم نمذجة الصيغ الزمنية والمخططات التحليلية وفق XML.




الشكل(٢-٥): البنى الهيكلية لأدوات IF

تعتمد لغة IF في نمذجتها لأي نظام على مبدأ الأتومات المحدد المنتهي Finite Automata حيث إن أي نظام يتكون من العناصر التالية:

- العمليات(processes): تأخذ شكل أتومات زمني وتتميز بأنها حتمية وذات منشأ ديناميكي.
- الاتصال اللامتزامن: يتم عبر قنوات اتصال خاصة إضافة إلى المتحولات المشتركة التي تفيد في تبادل المعلومات.
- بنى المعطيات المتبادلة ضمن النظام: يتم التصريح عن أنماط بياناتمعر ّفةمسبقاً (الأنماط الأساسية، المصفوفات، السجلات ) وأنماط بيانات مجردة (أنماط خاصيتُصر ّح عنها وفق تعليمات هذه اللغة) إضافة إلى تعريف متحولات توصيفية للنظام قدتُستخدم ضمن التعابير الحسابية.

### 1.7.2 الهيكلية البرمجية العامة للغة IF:

تتنو ّع تعليمات لغة IF لتتيح إمكانية نمذجة كافة الاحتياجات المختلفة التي قد تفرضها أنظمة الزمن الحقيقي، وقم ّ تصنيف هذه التعليمات حسب مدلولها البرمجي وفق التالي:

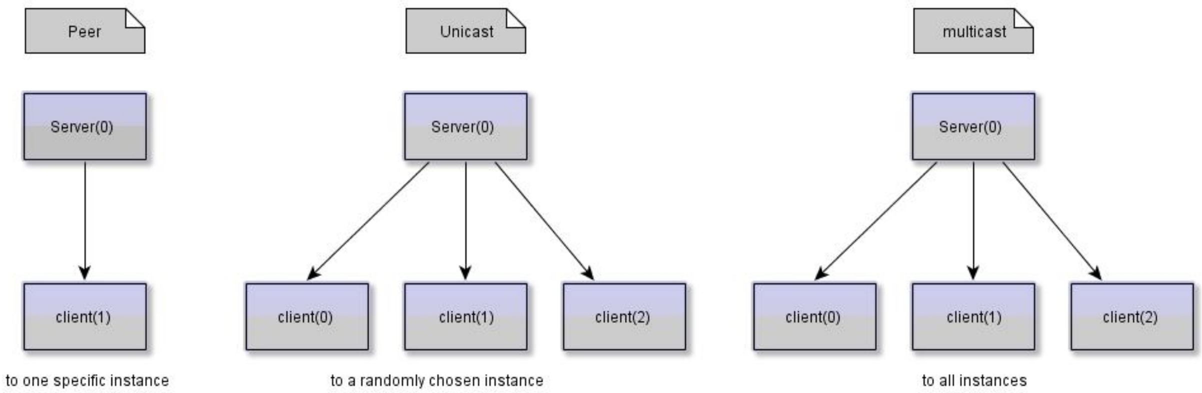
**هيكلية النظام**: يتكو ّن النظام ضمن لغة IF من عدد من الأغراضالفع ّالة (عمليات) التيتُنف ّذ على التفرع وتتفاعل فيما بينها بصورة غير متزامنة من خلال تبادل عدد من الرسائل (إشارات) ممررة عبرح ُجر اتصالات تخزينية (قنوات اتصال signalroute). التعليمات البرمجية التيتُحدد الهيكلية هي:

- النظام (system): يشمل على مكونات ديناميكية وأخرى ستاتيكية مثل المتحولات المشتركة وأنماط البيانات والمتحولات الثابتة والكتل البرمجية الخارجية الملحقة بالنظام (procedures).
- المُعالجة (process): تُعرّف العملية بأنها مجموعة الحالات المنتهية FSM التي يمكن أن تُنشأ وتُدمّر ديناميكياً أثناء تنفيذ النظام. تتميّز كل نسخة عن العملية (process instance) بمعرّف خاص به (Process Identifier (PID)) ورتل دخل يعمل على مبدأ FIFO تُخزّن ضمنه الرسائل القادمة (التي استقبلت ولم تُستهلك بعد من قبل هذه النسخة). يشمل توصيف العملية: اسمها، رقم النسخة المنشأة الذي يتحدّد وفق المكونات الموجودة في النظام)، قائمة المحولات الصورية (formal parameters) وغيرها من العناصر الأخرى التي قد تدخل في التوصيف (أهمها المتحولات المحلية التي تُحدد الذاكرة المحجوزة ومجموعة الحالات الانتقالية مع المناقلات المرتبطة بها التي تحدّد السلوك العام للعملية).
- قنوات الاتصال (Signalroute): تُحدّد القنوات التي تؤمن الاتصال بين العمليات ضمن النظام، وقد يحتوي النظام أكثر من نسخة واحدة للقناة تتميز كل منها بمعرّف خاص به ويمكن لهذه القنوات أن تُنشأ وتُدمّر ديناميكياً أثناء وقت التنفيذ. يشمل توصيف القناة: اسمها، رقم النسخة الحالية، الجهات الطرفية المحددة لمصدر ووجهة القناة (التي قد تكون عملية ما أو البيئة العامة للنظام environment)، إضافة إلى ذكر مجموعة الإشارات التي قد تعبر كل قناة. يمكننا أثناء توصيف سلوك القناة إضافة عدد من الخصائص والمزايا التي تجعل هذا التوصيف أكثر دقة وهذه الخصائص تُمثل الخيارات التالية:
  - ❖ تكديس الرسائل Queuing: تحدد أسلوب معالجة الرسائل المكدسة وتأخذ إحدى قيمتين:
    - Fifo: يتم تخديم الرسالة التي وردت للقناة أولاً حيث تُخزّن الرسائل وفق ترتيب ورودها ضمن رتل خاص للمحافظة على الأولويات فيما بينها.
    - Multiset: تُخزّن الرسائل وفق مجموعات متعددة ولا يوجد أي أولوية لترتيب ورود الرسالة المطلوب نقلها ضمن القناة.
  - ❖ وثوقية القناة Signalroute reliability وتأخذ إحدى القيمتين:
    - reliable: القناة موثوقة ولا يمكن فقد الرسائل ضمنها.
    - lossy: تدل على إمكانية فقد الرسالة أثناء نقلها ضمن قناة الاتصال.
  - ❖ تأخير الرسائل delaying: يتحدد التأخير الممكن للرسائل ضمن قناة الاتصال بإحدى القيم التالية:
    - Urgent: تدل على أن الرسالة عاجلة ينبغي تسليمها فورياً ولا يُقبل أي تأخير.
    - Delay[a,b]: أي رسالة تدخل القناة يجب أن تغادرها بعد a وحدة زمنية وخلال فترة لا تتجاوز b وحدة زمنية.

- Rate[a,b]: يتم تسليم الرسائل ضمن القناة بمعدل يتراوح وسطياً بين القيمتين [a,b].

❖ تسليم الرسائل delivering: تتحدد وفق الخيارات التالية الشكل(٢-٦):

- Peer: يتم تسليم الرسالة إلى نسخة العملية المحددة وفق الإشارة.
- Unicast: يتم تسليم الرسالة بشكل عشوائي إلى إحدى نسخ العملية المنفذة.
- Multicast: تُسلّم الرسالة إلى كافة نسخ العملية المنفذة الموجودة في نهاية قناة الاتصال.




**الشكل(٢-٦): استراتيجية تسليم الرسائل ضمن قنوات الاتصال**

- الإشارة Signal: تُستخدم الإشارة لتأمين التواصل بين العمليات حيث تُرسل الإشارات ضمن النظام أثناء مرحلة التنفيذ من نسخة عملية إلى أخرى. يشمل توصيف الإشارة: اسمها، وقائمة أنماط المتحولات المرفقة بها.

**سلوك النظام:** توصف أي عملية بتحديد الحالات الممكنة لها مع تعريف المناقلات والأحداث التي تؤدي إلى الانتقال بين هذه الحالات المختلفة وذلك من خلال:

- الحالة (state): تُمثل أحد أشكال السلوك الذي تبديه العملية أثناء تنفيذ النظام وقد تتضمن كل حالة على مناقلة تؤدي بها إلى الانتقال للحالة التالية كما قد تتضمن حالات فرعية أخرى، يمكن وفقها حفظ الإشارات غير المطلوبة ليتم معالجتها لاحقاً، كما يمكن تقييدها بشروط زمنية تُحدد فترة استمرارها، قد يُتبع اسم الحالة بواصفات تُحدّد طبيعة عملها فيما إذا كانت حالة ابتدائية أم حالة غير مستقرة unstable (لا يتم وفقها تنفيذ أي عمل).
- المناقلة (transition): تُمثّل المناقلة الاستجابة الموافقة لأحداث العملية وتتضمن على تعليمات بنى التحكم التسلسلية وتعليمات شرطية متتالية، قد تُرفق المناقلة بقيمة تُحدّد أولوية تنفيذها خلال زمن العملية. غالباً ما تنتهي المناقلة بالحدث Nextstate الذي يدل على انتقال التحكم إلى تلك الحالة أو قد تنتهي بالحدث Stop الذي يدل على تدمير نسخة الحالة، كما يمكن أن تتضمن التعليمات:
    - Provided expression: لاختبار القيود الزمنية المتاحة.

- When expression: لاختبار القيود غير الزمنية.
- Input signal-id: استقبال الإشارات الداخلة إلى النظام.

- الأحداث (action): يمكن تحديد الأحداث (الأفعال) التي يمكن إنجازها ضمن المناقلة بالتعليمات التالية:
  - Skip تجاهل تعليمة ما وهو حدث داخلي ضمن المناقلة، Informal الطباعة عبارة عن حدث قابل للمراقبة، Task إسناد قيم للمتحولات، set تفعيلعدّاد زمني من نمط clock، reset إعادة تهيئة قيم المتحولاتوالعدّاد الزمني، output إرسال إشارة ممكنة عبر قناة اتصال، call استدعاء إجراء برمجي، fork إنشاء ديناميكي لنسخة عملية/قناةوا إعادة قيمالمعرّف الخاص بها، kill تدمير عملية/قناة، نلاحظ بأن أحداث الاستدعاء call والإنشاء fork التي تعيد قيم محددة يمكن أن تترافق بعمليات إسناد للقيم المعادة.

### 2.7.2 تقنيات برمجية للتحقق من الصحة:

**تقنية Uppaal**: وهي عبارة عن أداة متكاملة [23] لنمذجة أنظمة الزمن الحقيقي ومحاكاتها والتحقق من صحتها وقطوّرت بالتعاون بين جامعات Uppsala و Aalborg وSweden.تُستخدم لنمذجة الأنظمة المكوّنة من مجموعة عمليات غير حتمية ذات بنى التحكم المنتهية والقيالموصّفة للزمن حيث يتم تبادل المعلومات فيما بينها عبر قنوات اتصال خاصة إضافة إلى قيم المتحولات المشتركة [24]، غالباً متاُطبّق هذه الأداة في مجال مراقبة الزمن الحقيقي وبشكل خاص ضمن بروتوكولات الاتصال ذات القيود الزمنية الحرجة. تتألف من ثلاثة أجزاء: لغة توصيف النظام،و المحاكي، وفاحص النموذج. تتضمن لغة التوصيف أوامر شرطية وبنى معطيات بسيطة (أعداد حقيقية، مصفوفة) وغيرها من التعليمات البرمجية المساعدة على توصيف سلوك نظام الزمن الحقيقي، المحاكيو هو أداةتُثبت صحة البرامج وتمكننا من اكتشاف الأخطاء التي قد تتخلله وذلك من خلال اختبار التنفيذ الديناميكي للنظام، أما فاحص النموذج فهوُقدّم السلوك الديناميكي الشامل من خلال اختبار قيم المتحولات الزمنية للنظام والمراحل المختلفة التي يمر بها خلال دورة حياته.

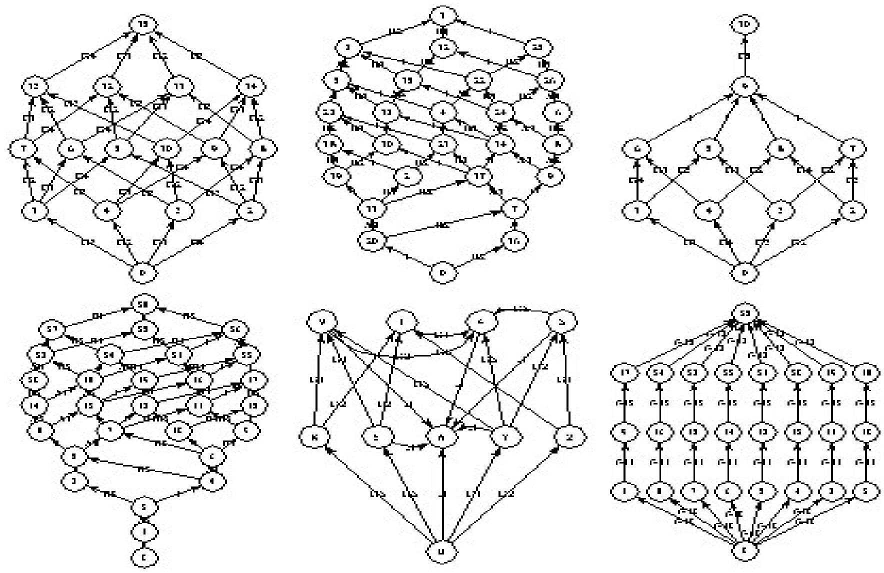
**تقنيــة CADP (Construction and Analysis of Distributed Processes)**: [25]دُعيت سابقاً بحزمة تطوير البرمجيات(Cæsar/Aldébaran)، وهي عبارة عن أداة لتصميم العمليات الموزعة وتحليلها. تم تطويرها من قبل فريق Vasy من InriaRhône-Alpes عبر عدة إصدارات حيث يتم في كل إصدار إضافة مكونات جديدة تلبي ما استجد من احتياجات برمجية، آخر إصداراتها النسخة CADP2006 "Edinburgh" التي تطورت خلال عامي 2007-2008 لتدعم منصات ٦٤ بت، تسمح CADP بتصميم البروتوكولات والأنظمة الموثوقة القابلة للاعتماد ويمكن استخدامها للتحقق من صحة برامج LOTOS كما تُمكننا من الحصول على المخططات الرسومية المقابلة لبرامج وتعليمات لغة IF (موضوع بحثنا). تحتوي

CADP العديد من العناصر البرمجية (أكثر من ٤٢ أداة و١٧عنصراً برمجي) المتكاملة مع بعضها البعض والمُصنّفة حسب الهدف البرمجي الذي تؤديه إلى: أدوات معالجة لـ LOTOS، أدوات معالجة مجال الحالة الضمني والصريح، أدوات فاحص الموديل(EVALUATOR, XTL)، الفاحص المكافئ ( BCG_MIN, BISIMULATOR)، الفاحص المرئي، أدوات الاختبار، أدوات تقييم الأداء، التحقق من صحة التطبيقات الموزعة، واجهات خاصة بالمستخدم،وأخيراً أدوات معالجة لعمليات الصيانة والتنصيب.تُتيح هذه الأدوات سهولة في الاستخدام وسرعة في الأداء من خلال الواجهة الرسومية Eucalyptus واللغة البرمجية النصية Svl المتاحة التي تقوم كلما اقتضت الحاجة بتحويل صيغة الملف الناتج عن هذه الأداة إلى الصيغة المناسبة للبيئة البرمجية الهدف كما تُزوّد بمجموعة من الأوامر النصية الملائمة لكافة الأغراض البرمجية المطلوبة. تتكون أدوات CADP التي تدعم معالجة LOTOS من المكونات التالية:

- Cæsar.adt: وهو مترجم يحوّل أنماط بيانات LOTOS التجريدية إلى أنماط وتوابع وفق لغة C، تتضمن عملية التحويل هذه تقنيات ترجمة النماذج المتطابقة والتعرّف على أتومات الأنماط المستخدمة.
- Cæsar: عبارة عن مترجم يحوّل بيانات LOTOS إلى لغة C بغرض تشكيل النماذج الأولية السريعة المعالجة وإجراء عمليات الاختبار، أو يحوّلها إلى مخطط بياني منتهي الحالات بغرض تطبيق عمليات التحقق، تتم عملية التحويل هذه وفق عدة خطوات مماثلة لخطوات بناء شبكة Petri-net [26] المكوّنة من متحولات نمطية وانتقالات أساسية إضافة إلى مزايا أخرى خاصة بمعالجة المعطيات، كما أن هناك الأداة CAESAR.INDENT والأداة CAESAR.BDD التي تُتيح تحليل شبكات Petri-nets الهيكلية من خلال مخططات اتخاذ القرار.
- Open/Cæsar: تُعدّ هذه الأداة البرمجية إحدى أدوات معالجة الحالة تساعد في دعم تقنيات استعراض المخطط البياني من خلال ما تقوم به من مهام كالمحاكاة والتحقق من الصحة وعمليات الاختبار وهي تُعد نواة أساسية للأداة CADP بما تُتيحه من اتصال للأدوات غرضية البرمجة مع الأدوات غرضية النمذجة.

أما بالنسبة للعناصر البرمجية BCG (Binary-Coded Graphs) التي تُعالج فاحص الموديل فهي:

- BCG_DRAW: تُزوّد بتمثيل ثنائي وثلاثي البعد مع عرض ديناميكي تلقائي مباشر للحالات والانتقالات المكافئة لها وفق مخططات الحالة الشاملة (الشكل ٢-٧).
- BCG_EDIT: محرر تفاعلي يسمح بالتعديلات الرسومية اليدوية المباشرة على المخططات البيانية المولّدة وفق الأداة السابقة.



**الشكل(٢-7): مخطط الحالات البياني وفق CADP**

كما أن هناك العديد من العناصر البرمجية الأخرى متنوعة الاستخدام مدعومة من قبل CADP (لسنا في مجال إحصائها). يمكن تنفيذ أداة CADP ضمن العديد من الأنظمة ومنصات العمل نذكر منها:

- محطات Sun (معالجات Sparc) القابلة للتنفيذ ضمن Solaris10 (موديل ٣٢ بت أو ٦٤ بت) مع مترجم خاص بلغة C (مترجم "Sun Studio 11 "cc أو "GNU "gcc) وبرمجية Ghostview.
- حواسيب شخصية (معالجات Intel/AMD x86) تعمل على نظام تشغيل Linux وفق 32 بت ضمن النسخ (kernel version >= 2.4, glibc version >= 2.3).
- مخدّمات (معالجات AMD64 أو Intel EM64T) التي تعمل على نظام Linux وفق 64 بت.
- مخدّمات (Intel Itanium) تُنفّذ على نظام تشغيل Linux وفق 64 بت.
- حواسيب شخصية (معالجات i386) قابلة للعمل ضمن نظام تشغيل Windows 2000 و XP أو نظام Vista مع دعم الأداة Cygwin أو برمجية Ghostscript/Gsview.
- حواسيب Mac (PowerPC) التي تعمل ضمن Mac OS X10.2 أو النسخ الأحدث مثل X11 إضافة إلى برمجية Ghostscript/Gsview.
- حواسيب Mac (معالجات Intel) التي تعمل ضمن Mac OS X10.4 أو النسخ الأحدث (X11) إضافة إلى برمجية Ghostscript/Gsview.

## 8.2 أدوات برمجية داعمة:

هناك بعض الأدوات البرمجية التي تم استخدامها بشكل جزئي أثناء هذا البحث لتسهيل العمل ودعمه ومن هذه الأدوات هي:

### 1.8.2 أداة LACATRE:

تُعدّ الـ LACATRE [27] لغة رسومية تستخدم لتصميم وتطوير تطبيقات أنظمة الزمن الحقيقي متعددة المهام، كما تسمح بالتعبير عن السلوك الديناميكي وأشكال العلاقات بين مكونات نظام الزمن الحقيقي والأنظمة المتضمنة. لهذه اللغة شكلان رسومي ونصي و تتمتع أيضاً بتوليد الشيفرة البرمجية بلغة الـ C مع نظام التشغيل الزمن الحقيقي iRMX المقابلة للمخطط الرسومي وبشكل أوتوماتيكي.

### 2.8.2 أداة Cygwin:

تُعد Cygwin [28] حزمة برمجية مفتوحة المصدر تزودنا ببيئة عمل افتراضية لنظام التشغيل Linux ضمن نظام التشغيل windows مما يجعل معظم تعليمات Linux قابلة للتطبيق المباشر. تم الاعتماد على هذه الأداة لتوليد ملف المحاكاة الشامل المقابل لبرنامج لغة IF (سيتم توضيح كيفية القيام بذلك لاحقاً)، كما يمكن العودة إلى الملحق /III/ لمعلومات إضافية عن هذه الخطوة.

## 9.2 الخاتمة:

تعرفنا في هذا الفصل على أنظمة الزمن الحقيقي ومتطلباتها ومواصفاتها الخاصة والقيود الزمنية التي قدتُ فُرض عليها، ووضحنا أهمية التحقق من صحة الشيفرة (code) البرمجية الخاصة بنظام زمن حقيقي قبل وضعه في مرحلة الاستثمار ووجدنا أن الخيار الوحيد الذي يُمكّننا من التحقق من صحة هذه الأنظمة والتأكد من قيودها الزمنية المفروضة عليها هو تحويل تعليمات نظم الزمن الحقيقي إلى لغة أخرى تتيح لنا ذلك هذه اللغة هي لغة IF التي تمتلك ميزات خاصة تدعم عملية التحقق المطلوبة. أي أن هناك حاجة ماسة لوجود مترجم يساعد في تحويل البرامج المكتوبة وفق لغة C/VxWorks إلى التعليمات المقابلة لها وفق لغة IF، ليُتاح لنا التحقق من صحة شيفرتها البرمجية [29] وتحديد مدى مطابقتها للمواصفات المطلوبة. لتحقيق ذلك علينا البحث في طرق وتقنيات بناء المترجمات بالشكل الذي يساعدنا على بناء مترجمنا المنشود واستخدامه ضمن تطبيقات نظم الزمن الحقيقي وهذا هو موضوع الفصل القادم.

# 3

# الفصل الثالث

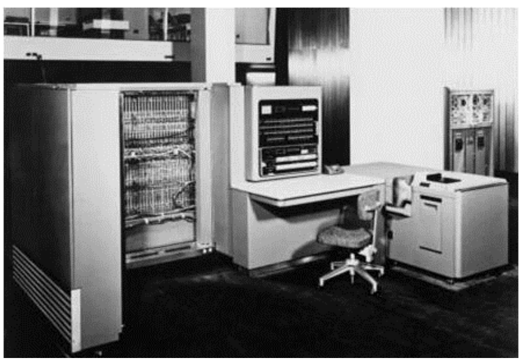
# المترجمات وأدواتها

1.3 مقدمة.
2.3 تعريف المترجم.
3.3 مراحل بناء المترجم.
4.3 بناء المحلل اللفظي والتركيبي.
5.3 الخاتمة.

# المترجمات وأدواتها

## 1.3 مقدمة:

طوّرتشركة IBM عام ١٩٥٣ الآلة الالكترونية الأولى المعالجة للبيانات الشكل(3-1) EDPM (Electronic Data Processing Machine)لتُستخدم كآلة حاسبة معتمدة في برمجتها على لغة التجميع Assembly. غير أن John Backus في العام نفسه طرح فكرة تسريع البرمجيات ودعا إلى الاستغناء عن هذه اللغةنظراً لصعوبة تعليماتها وقربها من لغة الآلة (اللغة الأم native code)، فقام بتطوير أول مفسر في التاريخ إلا أنه كان بطيئاً جداً فمعالجة التعليمات وفقه استغرقت فترات زمنية تعادل عشرات المرات للوقت المستغرقسابقاً وفق لغة Assembly، وفي عام ١٩٥٤قدّم بحث متكامل عن نظام FORTRAN يقوم بترجمة الصيغ الرياضية المعتمدة في IBM، ثمطُرحت FORTRAN كلغة برمجة عام ١٩٥٦ لتتيح إمكانية إنجاز المشاريع البرمجية خلال ساعتين بعد أن كانت تستغرق أسبوعين، ما لبثت هذه اللغة أن انتشرت عام ١٩٥٨ وأصبحت معتمدة في برمجة غالبية المشاريع.



الشكل(3-1): الآلة الحاسوبية الأولى

تطورت لغات البرمجة بعد ذلككثيراً وظهرت العديد من اللغاتوصُنّفت إلى:

- لغات منخفضة المستوى (Low Level Language): لغة التجميع (assembly) ولغة الآلة (machine code).
- لغات عالية المستوى (High Level Language): تمتاز بوضوحها ودقتها وأنها قريبة للفهم وأسهل للتذكر من قبل المبرمج.و أصبح تنفيذ أي برنامج مكتوب وفق لغة عالية المستوى يحتاج إلى عدد من مراحل المعالجة المختلفة التيتُطبّق انطلاقاً من الشيفرة المصدرية للبرنامجوصولاً إلى لغة الآلة القريبة للغة المعالج.

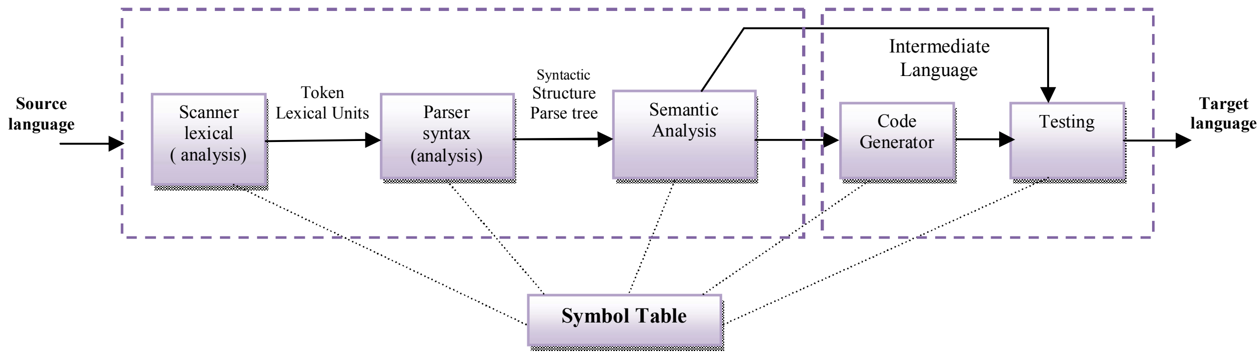
## 2.3 تعريف المترجم:

هناك العديد من التقنيات المعتمدة لترجمة البرامج المكتوبة وفق لغات البرمجة المختلفة:

- المفسِّر (Interpreter): يتحقق قواعدياً من صحة الشيفرة البرمجية المكوِّنة للبرنامج تعليمة تلو الأخرى فإن كانت صحيحة تم تنفيذها مباشرة والانتقال إلى التعليمة التالية وإلا يتوقف عن متابعة عمله إلى أن يتم تصحيح الخطأ المرتكب، فهو بالتالي لا يستغرق وقت معالجة مسبقة لعملية التنفيذ كما أنه لا يحتاج إلى أي تعديلات إضافية لينفذ على منصة عمل مختلفة عن بيئة التنفيذ، ويعتمد المفسر في العديد من لغات البرمجة مثل Visual Basic.
- المترجم (Compiler): يتعامل مع البرنامج ككتلة واحدة بحيث لا يتم الانتقال إلى مرحلة التنفيذ النهائي إلا إذا كان البرنامج وبشكل كامل خالياً من الأخطاء القواعدية، وهذا قد يستغرق وقتاً إضافياً للقيام بعملية المعالجة الشاملة ولكنه يجعل البرنامج ينفذ وبسرعة تفوق (٢٠–2) مرة سرعة التنفيذ وفق المفسر، ويطبق المترجم في العديد من لغات البرمجة عالية المستوى مثل Visual Studio.net و++C/C و#C.
- تستخدم بعض اللغات الحديثة كلاً من المترجم والمفسر في ترجمة تعليماتها مثل لغة Java التي تعتمد على المترجم في تحويل تعليماتها إلى لغة التجميع الافتراضية والقابلة للتفسير على أي حاسوب لاحقاً.
- تعتمد بعض اللغات البرمجية مثل (LISP) في بعض محرراتها على المترجم و في بعضها الآخر على المفسر، غير أن التطورات الحديثة جعلت المترجمات أكثر شيوعاً واستخداماً من قبل كافة اللغات البرمجية الحديثة.

## 3.3 مراحل بناء المترجم:

إن مترجم أي لغة هو عبارة عن برنامج يقوم بقراءة تعليمات لغة الدخل عالية المستوى ويحوِّلها إلى تعليمات لغة أخرى منخفضة المستوى، وكما هو معروف فإن المترجم [30] متعدد المراحل (compiler Multi phases) (الشكل 3–2)،نظراً لوجود بعض القضايا البرمجية التي لا يمكن معالجتها مباشرة في مرحلة واحدة وإنما تبقى دون إجابة حتى مرحلة لاحقة (كاستدعاء معرَّف اسم تابع سيتم التصريح عنه لاحقاً في نهاية البرنامج) أو عدم وجود ذاكرة كافية للتعامل مع كافة معلومات البرنامج خلال مرحلة واحدة (نستطيع حفظ النتائج الوسيطة لمرحلة ما ضمن ملف ثم قراءته في المرحلة التالية). يتألف المترجم من: التحليل اللفظي،و التحليل التركيبي، وتحليل المعاني، وتوليد برنامج الهدف وأخيراً اختبار البرنامج للتحقق من صحة مطابقته للهدف البرمجي المنشود.




**الشكل(3−2): مراحل بناء المترجم**

- **مرحلة التحليل اللفظي (Lexical Analysis):** تأخذ هذه المرحلة دخلها من تعليمات برنامج المصدر المطلوب ترجمته ثم تقوم بتشكيل سلاسل المقاطع Token المقابلة له والمؤلفة من محارف الأبجدية الخاصة بلغة الدخل، يتم التعامل مع كافة أحرف الأبجدية بأسلوب موحّد إذا كانت اللغة غير حساسة لوضعية الأحرف (لا تُميّز بين الأحرف الكبيرة والصغيرة) تُصنّف عادة هذه المقاطع وفق أنماط محددة: معرّفات، كلمات مفتاحية، ثوابت رقمية، ثوابت محرفية، معاملات. إضافة إلى معالجة المقاطع غير المحرفية Non-tokens التي قد توجد في البرنامج: كالتعليق، الموجهات التي تُضمِّن المكتبات المعالجة مسبقاً، الفراغات، علامات الجدولة وعلامة السطر الجديد.
- **مرحلة التحليل التركيبي (Syntax Analysis):** هي عملية إنشاء شجرة تحليل خاصة بالبرنامج المُترجَم والمطابقة لقواعد لغته ليتم وفقها اكتشاف الأخطاء القواعدية التي تم اختراقها ضمن تعليمات البرنامج. تتحدد كل لغة بأبجديتها وبمجموعة القواعد التي تضبط برامجها وللقيام بالتحليل القواعدي الصحيح لابدّ من بناء شجرة تحليل قواعدي مقابلة لبرامج لغة الدخل إذ لا يفيد المسح الخطي للبرامج في تحقيق ذلك.
- **مرحلة تحليل المعاني (semantic Analysis):** يتم في هذه المرحلة الاهتمام بمعاني البرنامج أي التحقق فيما إذا كانت الأجزاء المختلفة من البرنامج تؤدي معنى متكاملاً مع بعضها البعض يتوافق مع الهدف المطلوب. يتم أيضاً ضمن هذه المرحلة التأكد فيما إذا تم التصريح عن كافة متحولات البرنامج ومدى توافق المعاملات مع أنماطها التصريحية.
- **مرحلة توليد الشيفرة البرمجية (Code Generator):** يتم كتابة الشيفرة البرمجية والعمل على تحسينها (Optimizer) بالشكل الذي يضمن اختزال التعليمات (حجم الشيفرة أصغر)، وسرعة المعالجة والترجمة، أداءً عالياً للبرمجية، وقابلية القراءة والتنقيح، وشيفرة آمنة مفهومة من قبل المبرمج وأقل قابلية للفهم من قبل الآخرين.
- **مرحلة الاختبار (Testing):** يتم وفقها التحقق من صحة التعليمات البرمجية والأنماط (Type Checking) والتأكد من مدى مطابقتها لمتطلبات النظام الهدف.

تتضمن كل مرحلة من المراحل السابقة جدول معطيات خاصاً بها يحتوي معلومات عن نمط ومواصفات كل معرف مستخدم ضمن البرنامج، يتم تحديد بياناته خلال مرحلة التحليل اللفظي ويتعهد المترجم لاحقاً إدارته كمهمة أساسية له، إضافة إلى ضرورة القيام بمعالجة الأخطاء التي تظهر في كل مرحلة بشكل مباشر.

## 4.3 بناء المحلل اللفظي والتركيبي:

للقيام بمرحلتي التحليل اللفظي والتركيبي بدأنا أولاً بتحليل لغة الدخل للمترجم وبنينا القواعد بصيغة BNF(Backus-Naur Form)، ومن ثم انطلاقاً من هذه القواعد قمنا باستخدام أدوات التحليل اللفظي والتركيبي المزودة مع نظام تشغيل linux (Lex and yacc) [31] لما لها من سهولة في الاستخدام وسرعة في العمل إذ أنها تُقدِّم قوالب برمجية جاهزة للاستخدام المباشر بأسلوب يُسهّل عملية تحقيق هاتين المرحلتين ويوفر الكثير من الوقت والجهد الذي قد يبذله المبرمج إذا لجأ لإنجازهما بالطرق التقليدية.

### 1.4.3 قواعد BNF للغة C/VxWorks:

تُعدّ النماذج القواعدية من أهم المفاهيم التي ساعدت في توصيف لغات البرمجة وتصميم المترجمات، أول من قام بتعريف مفاهيم هذه القواعد وأسلوب كتابتها هو John Backus و Peter Naur حيث حُدِّدت البنى القواعدية لأي لغة على أنها بنى تعاودية تُوصّف وفق القواعد خارج السياق CFG (Context-Free Grammars)، ولقد قمنا بالاعتماد على هذه المفاهيم لدى تعريفنا لقواعد لغة C التي تدعم كتابة البرامج ضمن نظام التشغيل VxWorks مع الأخذ بعين الاعتبار كافة تعليمات الزمن الحقيقي وذلك لاستخدامها خلال مرحلتي التحليل اللفظي والتركيبي. على سبيل المثال يُعبّر الجدول التالي عن تعريف مهمة Task في لغة VxWorks وقاعدة الـ BNF الموافقة لها.

| Instruction in C/VxWorks | BNF Grammar |
|---|---|
| int taskID= taskSpawn( char *name, int priority, int options, int stackSize, FUNCPTR entryPt, int arg1, int arg2, int arg3, int arg4, int arg5, int arg6, int arg7, int arg8, int arg9, int arg10) | **task_declaration** :<br>Int_variable "=" "taskSpawn" "(" Char_Variable "," task_var ","<br>task_var "," Int_variable "," Func_RTS_Variable "," Int_variable ","<br>Int_variable "," Int_variable "," Int_variable "," Int_variable ","<br>Int_variable "," Int_variable "," Int_variable "," Int_variable ","<br>Int_variable ")" |

**الجدول(٣-١): صيغة BNF المقابلة لتعليمة taskSpawn**

بأسلوب مماثل قمنا بتحديد قواعد BNF التي تُوصّف البرامج المكتوبة وفق لغة C/VxWorks إضافة إلى التوابع البرمجية المُستخدمة ضمن نظام التشغيل VxWorks والمتضمنة في المكتبات الجاهزة (أشرنا إليها في الفصل الثاني) ليتم الاعتماد عليها في كتابة ملف التحليل التركيبي، ولقد قمنا بذكر هذه القواعد بتفصيل أكثر ضمن الملحق /I/.

### 2.4.3 تقنية Lex (A Lexical Analyzer Generator):

وهي تقنية [32] تستخدم في التحليل اللفظي تقوم بتحويل برنامج الدخل إلى سلاسل تدعى tokens باستخدام التعابير المنتظمة (regular expression)وتُولّد تلقائياً مايُدعى بـ (lexer أو scanner) ليتم تقديمها إلى محلل التراكيب syntax analyzer بصورة ملف (*.l file) لمعالجتها والتعرف عليها (مثل عملية تقسيم ملف نصي إلى كلمات مستقلة) .

❖ الصيغة التوصيفية لملف الـ lex:

| | |
|---|---|
| %{<br>< C global variables, prototypes, comments ><br>%} | يُضمّن هذا الجزء ضمن ملف lex.yy.c |
| [DEFINITION SECTION] | البدائل الخاصة بهذا الملف وحالات البداية تُنسخ ضمن ملف lex.yy.c |
| %%<br>[RULES SECTION] | تُحدد الأفعال المتخذة لكل token. |
| %%<br>< C auxiliary subroutines> | التعليمات المعرفة من قبل المستخدم، كاستدعاء التابع yylex |

❖يُكتب قسم القواعد الخاصة بهذا الملف بالشكل:

```
%% [قسم القواعد]
<النموذج> {<الفعل الذي يجب اتخاذه لدى حدوث المطابقة >}
<النموذج> {<الفعل الذي يجب اتخاذه لدى حدوث المطابقة >} ... %%
```

توصّف النماذج ضمن هذا القسم وفق التعابير المنتظمة (الجدول ٣-2) معتمدة في تمثيلها الأساسيات التالية:

- (.): تدل على مطابقة أي حرف مفرد باستثناء رمز (\n) الدال على سطر جديد.
- (*): تدل على مطابقة أو أكثر لنسخ التعبير المنتظم السابق أو عدم وجود أي مطابقة.
- (+): تدل على مطابقة أو أكثر لنسخ التعبير المنتظم السابق.
- (?): تدل على عدم المطابقة أو مطابقة واحدة فقط للتعبير السابق.
- (|): تدل على إمكانية مطابقة التعبير السابق أو اللاحق.
- ([]): تحديد صف من الأحرف.
- (()): تغليف التعابير المنتظمة المختلفة ضمن تعبير جديد.
- "..." : مطابقة تامة لأي رمز موجود بين إشارتي الاقتباس العلوية.

فالنموذج المعبّر عن كتابة عبارة ما لدى مصادفة تسلسل محرفي من الأحرف كبيرة أو صغيرة يوصف بالقاعدة: %% [A-Za-z]+ { printf("this is a word"); } %%

❖ تتم المطابقة وفق ملف الـ Lex لأبعد حد ممكن من عدد المحارف، وفي حال وجود احتمال لأكثر من مقطعين بالطول نفسه فإن المقطع للتعبير المنتظم المعرّف أولاً ستُأخذ المطابقة لصالحه.

| الهدف المطلوب | التعبير المنتظم المقابل |
|---|---|
| معرّف نهاية جملة | "." \| "?" \| ! or ["."" ?"!] |
| التعليق ضمن لغة C++ | "//".* |
| الفراغ ضمن المقاطع | [ \t]+ |
| تسلسل ما من الرموز والمحارف المختلفة | ([ \t]+ \| [a-zA-Z]+)+("." \|"?" \|!) |
| ورود رقم صحيح دون إشارة، مثل 12345. | [1-9][0-9]* |
| ورود رقم صحيح ذي إشارة، مثل 12- أو 12+ | ["-"+]?[1-9][0-9]* |
| عدد حقيقي مثل 1.2345 | [0-9]*"."[0-9]+ |

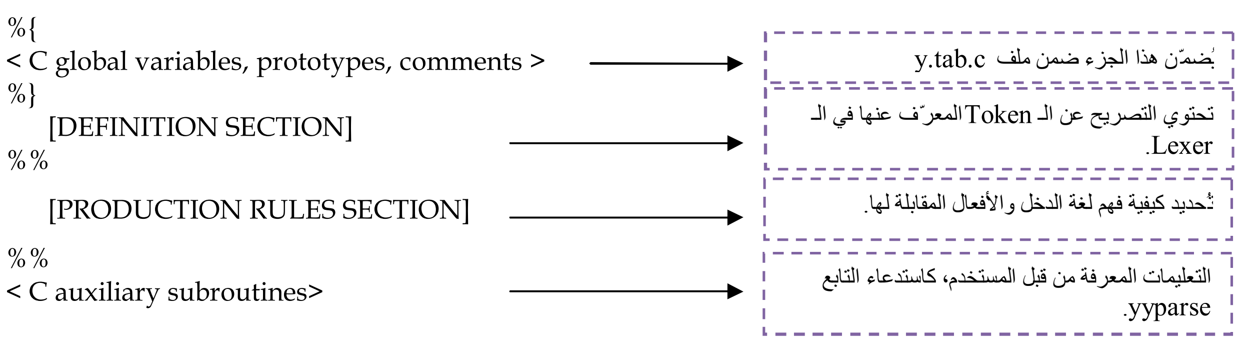
**الجدول(٣-2): صيغ لتعابير منتظمة**

يتم خلال هذه المرحلة تجاهل الفراغات والأسطر الفارغة، تجاهل التعليقات comments، التعرف على الكلمات المحجوزة (Reserved Words) ضمن لغة C إضافة إلى التوابع الخاصة بعناصر الزمن الحقيقي المعالجة ضمن مترجمنا،إضافة إلى التعرّف على الرموز المحجوزة مثل (+) للجمع (-) للطرح والتعرف على الأرقام إضافة إلى السلاسل النصية.

### 3.4.3 تقنية Yacc (Yet Another Compiler - Compiler):

وهي تقنية [32] تستخدم كمحلل للقواعد من أجل التأكد من البنية التركيبية للبرنامج وتأخذ بياناتها من خرج المحلل اللفظي (الـ tokens التي حصلنا عليها من الأداة السابقة) وتحللها بشكلها القواعدي المحدد للغة.

❖ الصيغة التوصيفية لملف الـ YAcc:




❖ يمكننا ضمن القسم الأول من هذا الملف التعامل مع كلمات مفتاحية ذات دلالات خاصة بهذا الملف:

- '%start` تُحدد رمز بداية القواعد.
- '%union` يُصرّح عن مجموعة من أنماط المعطيات المختلفة التي قد تدل لاحقاً على قيمٍ معان .
- '%token` يُصرّح عن رمز نهاية (اسم Token) دون ذكر أي أولوية أو ترابط مع رموز أخرى.
- '%type` التصريح عن نمط من القيم للرموز غير المنتهية.
- '%left` يُصرّح عن رمز نهاية ذي أولوية تنفيذ من اليسار. مثلاً نكتب (%left '-' '+').

- ' right%' يُصرّح عن رمز نهاية ذي أولوية تنفيذ من اليمين.مثلاً نكتب (' ^ ' right%).
- 'nonassoc%' يُصرّح عن رمز نهاية لا يمتلك أولوية، بحيث يؤدي إلحاقه بأولوية معينة إلى Syntax Error.

❖ يُكتب قسم القواعد الخاصة بهذا الملف بالشكل:

```
 %%
production : symbol1 symbol2 …{ action }
           | symbol3 symbol4 …{ action }
           | …
production: symbol1 symbol2     { action }  | …  %%
```

❖ على سبيل المثال لدينا القاعدة التالية:

```
  %% statement : expression ;               { printf (" = %g\n", $1); }
     expression : expression '+' expression { $$ = $1 + $3; }
           | expression '-' expression      { $$ = $1 - $3; }
           | NUMBER ;                       { $$ = $1; }    %%
```

يدل $1 على الحد الأول، ويدل $3 على الحد الثالث بينما يُعبّر $$ عن قمة المكدس المكافئ للقاعدة بعد الاختزال. قد تظهر بعض التعارضات والتناقضات أثناء كتابتنا لقواعد ملف Yacc وهي من نمطين:

- shift/reduce: مثل القاعدة:

```
stmt: IF expr stmt | IF expr stmt ELSE stmt;
```

في هذه الحالة يقوم Yacc بتقصّي أكبر مطابقة ممكنة أي يعتمد على إزاحة القاعدة لآخر احتمال ممكن.

- reduce/reduce: مثل القاعدة:

```
start : expr | stmt ;
expr : CONSTANT;
stmt : CONSTANT;
```

يقوم Yacc بحل هذه المشكلة باختياره القاعدة التي تُذكر أولاً (إنقاص القاعدة) لكن ذلك قد لا يكون صحيحاً دائماً، لذا فإن كتابتنا للقواعد بأسلوب واضح يجعلنا نحصل لاحقاً على نتائج أكثر دقة.

### 4.4.3 ملف التحليل التنفيذي النهائي:

بعد أن قمنا بكتابة ملف التحليل اللفظي (vxworkslex.l) وفق قواعد الـ Lex (الشكل 3-3) وملف التحليل التركيبي (vxworksyacc.y) وفق قواعد الـ Yacc، تم استخدام أداتي التحليل Flex و Bison لمعالجة هذين الملفين وإنتاج الملف التنفيذي للمحلل النهائي وذلك بتطبيق المعالجات التالية مع مراعاة ترتيب تنفيذها لأن بعض الملفات تعتمد على معلومات ناتجة عن خطوات سابقة (مثلاً يتم ضمن ملف التحليل اللفظي استدعاء الملف (vxworksyacc.tab.h) الناتج عن معالجة سابقة لملف التحليل التركيبي):

- يتم الحصول على الملف (vxworksyacc.tab.c) والملف (vxworksyacc.tab.h) الذي ينتج عن إضافة الخيار (d-) من خلال تنفيذ التعليمة: `> bison -d vxworksyacc.y`
- الحصول على الملف (lex.yy.c) بتنفيذ التعليمة: `> Flex vxworkslex.l`
- الحصول على الملف (vxworksyacc.tab.o) بتنفيذ التعليمة: `> gcc -c vxworksyacc.tab.c`

- الحصول على الملف (lex.yy.o) بتنفيذ التعليمة: > gcc -c lex.yy.c

- أما تنفيذ التعليمة التالية فيؤدي إلى إنتاج المحلل التنفيذي النهائي باسم اختياري (testvxworks):

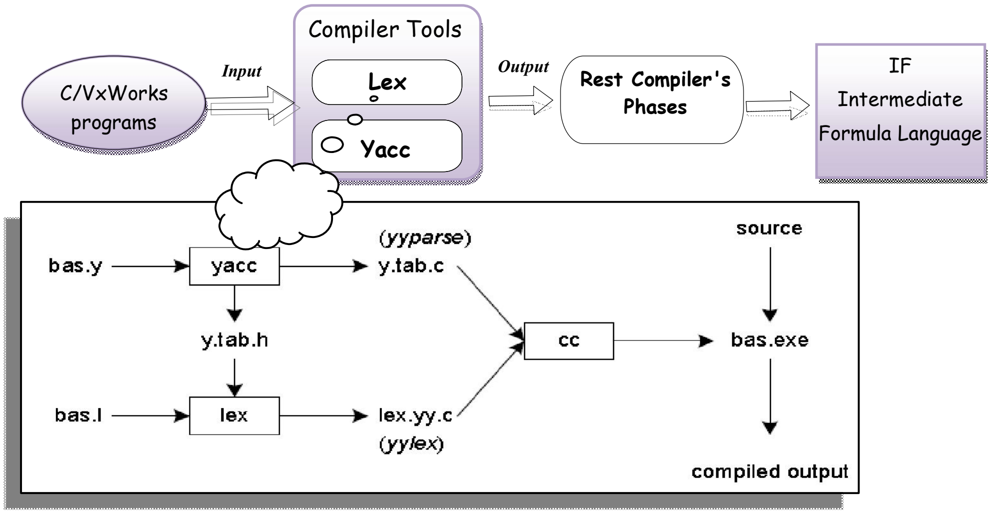
> gcc -o testvxworks vxworksyacc.tab.o lex.yy.o -lm –lfl

ولاختبار صحة نتائج هذا المحلل والتأكد من قدرته على اكتشاف تعليمات ملف الدخل التي لا تحترم قواعد لغة C/VxWorks يتم تنفيذ التعليمة التالية: > testvxworks exp.txt res.txt

يتم وفق هذه التعليمة (الشكل 3–4) تطبيق المحلل اللفظي والتركيبي على الملف النصي (exp.txt) وتوجيه نتائج التحليل إلى ملف الخرج (res.txt) لمقارنتها مع النتائج المنتظرة التي تم تحديدها لدى تنفيذ البرنامج نفسه ضمن بيئة Tornado.

```
%{                      /*  C-libraries and Token definitions */
#define YYSTYPE double
extern YYSTYPE yylval;
#include <string.h>
#include <stdlib.h>         /* for calling the function atoi */
#include <stdio.h>
#include " vxworksyacc.tab.h"
%}
        CHAR                    [a-z][a-z]*
        VARNAME                 [a-z][a-z0-9]*
        DIGIT                   [0-9][0-9]*
%%                      /*  TOKEN Definitions   */
        "<"                     {return(SML);               }
        "h"                     {return(EXT);               }
        ">"                     {return(BIG);               }
        "."                     {return(POINT);             }
        "#"                     {return(SHARP);             }
        ";"                     {return(SEMICOLON);         }
        "{"                     {return(LEFBRK);            }
        "}"                     {return(RIGBRK);            }
        ":="                    {return(ASSIGN);            }
        "+"                     {return(SUM);               }
        "include"               {return(include);           }
        "STATUS"                {return(STATUS);            }
        "taskDelete"            {return(taskDelete);        }
        "taskSpawn"             {return(taskSpawn);         }
        "taskResume"            {return(taskResume);        }
        "taskSuspend"           {return(taskSuspend);       }
        "taskDelay"             {return(taskDelay);         }
        "VOID"                  {return(VOID);              }
        "("                     {return(LEFARC);            }
        ")"                     {return(RIGARC);            }
        {CHAR}                  {return(LETTERS);           }
        {VARNAME}               {return(VARIABLENAME);      }
        {DIGIT}                 {yylval = atoi ((char *)yytext); return(VALUE);}
        [ \t\n]+                /* eat up whitespace */
        .                       { return(yytext[0]);        }
%%
```

**الشكل(3–3): جزء من تعليمات ملف التحليل اللفظي**




الشكل(3-4): بناء المترجم باستخدام Lex/Yacc

## 5.3 الخاتمة:

قمنا في هذا الفصل ببناء محلل خاص بمرحلتي التحليل اللفظي والتركيبي للغة C/VxWorks ووضحنا كيفية تعريف قواعد هذه اللغة وفق صيغة الـ BNF لكتابة ملفي التحليل اللذين تتم معالجتهما بأدوات تحليل الـ Flex والـ Bisonوأخيراً تم مناقشة كيفية التأكد من صحة نتائجهما وتكاملهمامعاً . بعد قيامنا بهاتين المرحلتين تأتي مرحلة تحليل المعانيللتعرّف على صحة دلالة التعليمات ضمن برنامج الدخل ومن ثم تليها مرحلة توليد الشيفرة البرمجية النهائية القابلة للتنفيذ ولتحقيق ذلك لابدّ من تحديد التقابلات بين لغة الدخل C/VxWorks ولغة الخرج IF بالشكل الذي يضمن لنا صحة النتائج النهائية.

هذا ما سنعالجه في الفصل القادم حيث سنوضح كيفية تحديد هذه التقابلات والأسلوب المعتمد للحصول على تطبيق نهائي قابل للاستخدام المباشر.

# ٤ الفصل الرابع

# الانتقال من C/VxWorks إلى لغة IF

# C/VXWORKS TO IF


1.4 مقدمة.

2.4 المقابلة بين عناصر C/VXWORKS وIF.

3.4 القواعد المتبعة في عملية المقابلة ومعالجة المشاكل البرمجية.

4.4 اختبار تكاملية المترجم.

5.4 الخاتمة.

# C/VXWORKS TO IF



## 1.4 مقدمة:

ذكرنا سابقاً عناصر الزمن الحقيقي وكيف أنها تتنوع بين عناصر قابلة للبرمجة وأخرى قابلة للتهيئة، ووضحنا مفاهيمها ووظائفها والتوابع التي تُنظم عمل كل منها، كما ناقشنا ضرورة بناء مترجم يقوم بتحويل تعليمات لغة C/VxWorks إلى لغة IF كخطوة أساسية لهذا البحث الذي يهدف إلى التحقق من صحة تطبيقات الزمن الحقيقي قبل استثمارها تفادياً لأي أخطاء فادحة مميتة للنظام. لقد قمنا خلال الفصل السابق بإنجاز مرحلتي التحليل اللفظي والتركيبي لهذا لمترجم وسنتابع الآن بقية مراحل بناء المترجم لنناقش كيفية القيام بمرحلة توليد الشيفرة البرمجية (Code Generation) القابلة للتنفيذ المباشر وما يتطلبه ذلك من مهام ضرورية. إن عملية توليد الشيفرة البرمجية الخاصة بمترجمنا تقتضي أولاً تحديد التعليمات المتقابلة بين اللغتين التي ستُعتمد في كتابة الشيفرة والتحقق من صحة أدائها البرمجي بالشكل الذي يضمن الحصول على نتائج صحيحة وسليمة، وهذا ما سنقوم بعرضه حالياً .

## 2.4 المقابلة بين عناصر C/Vxworks وIF:

تُعدّ لغة IF لغة نصية يتكون فيها النظام من مجموعة من العمليات التي تُمثّل بدورها مخطط الحالة المنتهي (Finite State Machine) FSM، وللقيام بعملية المقابلة هذه تم تحديد وإحصاء كافة عناصر الزمن الحقيقي ضمن لغة C/VxWorks ودراسة سلوكية كل عنصر من خلال الأداء البرمجي للتوابع الخاصة به والعمل على إيجاد التعليمات المكافئة له وفق لغة IF ومن ثم قمنا بتمثيل هذه التعليمات المتقابلة الناتجة بمخطط الأتومات [33] وفق الأشكال البيانية التي ستُذكر لاحقاً ضمن هذه الفقرة والتي هي عبارة عن أشكال توضيحية لتعليمات المقابلة الفعلية المحددة من قبلنا وهي جزء أساسي من بحثنا وقد قمنا برسمها بهذا الأسلوب لإيضاح تعليمات التقابل المطلوبة التي تُعدّ خطوة جوهرية من عملنا لبناء مترجمنا الهدف، وقد اعتمدنا على الأتومات في التمثيل البياني للتعليمات المتقابلة بين اللغتين نظراً لسهولة إظهار هذه التعليمات وتوضيحها وفقه إذ أن لغة IF لا تدعم أي تمثيل بياني لمخطط حالات النظام المعالج.

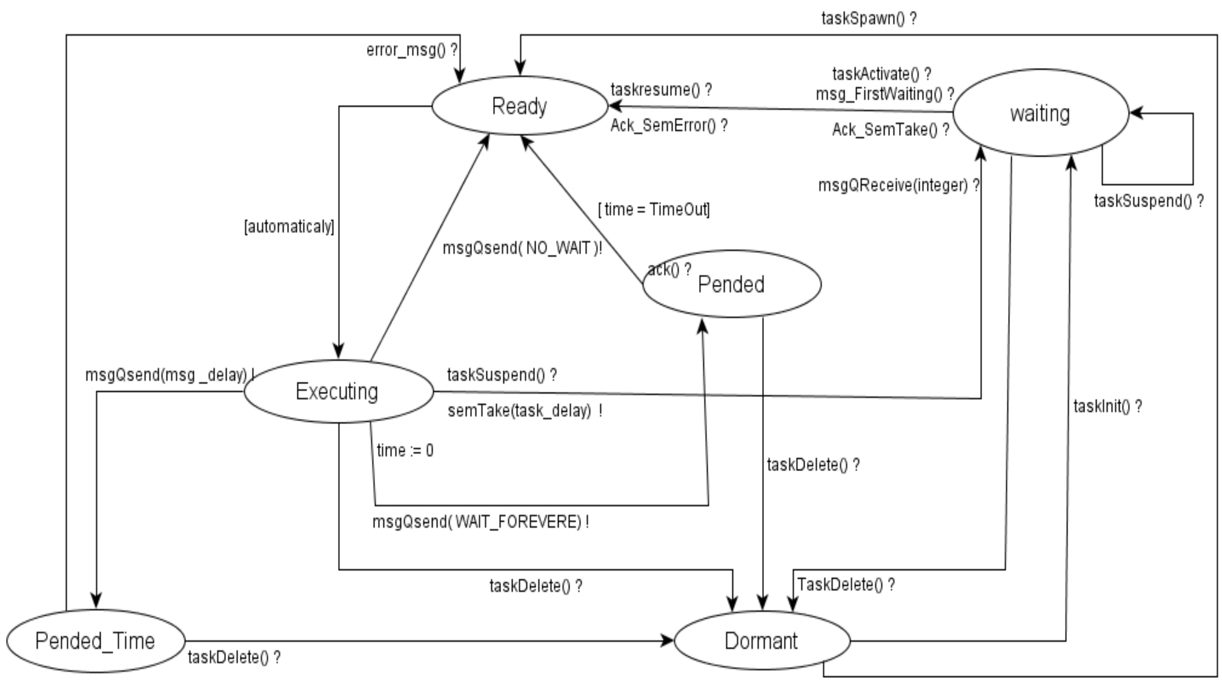
أثناء المقابلة تمّ اعتبار كل عنصر على أنه عملية تتضمن عدداً من الحالات المختلفة التي تعتمد على الإشارات المتبادلة فيما بينها لإنجاز العمل المطلوب من النظام، وهذه الإشارات تُمثّل التوابع البرمجية المقابلة للعنصر المدروس ضمن نظام VxWorks.

يمكننا الآن عرض وشرح كل عنصر مع أسلوب تحليلنا لأدائه وكيف تم مقابلة سلوكه بتعليمات لغة IF، كما يمكن العودة إلى الملحق /II/ من أجل تفاصيل برمجية أكثر عن تعليمات المقابلة الناتجة.

### 1.2.4 المهمة Task:

قمنا بدراسة سلوكية عنصر المهمة مع تحديد ما يقابل أداء توابعه من تعليمات مصاغة وفق لغة IF على شكل إشارة خرج مرسلة أو إشارة دخل مستقبلةو الحالة المتوقعة للعملية لدى إرسالها أو استقبالها لهذه الإشارات وما ينجم عنها من انتقالات بين حالات النظام. علماً أنه أثناء مقابلة التعليمات تم مراعاة المتحولات الوسيطة للتابع المعالج بحيث نحافظ فقط على المتحولات المحددة للمهمة، فعلى سبيل المثال يتضمن التابع taskSpawn عدداً من المتحولات الواصفة للمهمة المراد إنشاؤها وتفعيلها هذه المتحولات هي: taskname,priority,options,stackSize,EntryPointFunction,arguments. نلاحظ لدى مقابلة هذا التابع أن المتحولين taskname,EntryPointFunction هما أكثر المتحولات أهمية لذا يتم الاعتماد عليهما في مرحلة الترجمة وفي تحديد تعليمات العملية المكافئة للمهمة المدروسة، أما باقي المتحولات فلها معنى دلالي ضمن C/VxWorks ولكن ليس لها أي أثر برمجي وفق شيفرة IF.

أخيراً تم تمثيل المهمة وفق لغة IF بعملية (الشكل ٤-١) تتضمن عدداً من الحالات المختلفة،تُعتبر حالة dormant الحالة الابتدائية للمهمة التي تعني عدم إنشاء المهمة بعد، ولا تصبح المهمة في حالة جاهزية (ready) إلا بعد إنشائها الذي عبرنا عنه باستقبال إشارة taskSpawn. أما استقبال الإشارة taskInit فلا يُعتبر كافياً لتفعيل المهمة وإنما لابدّ من وصول الإشارة taskActivate لتنتقل المهمة إلى حالة ready. تبدأ المهمة عملها التفاعلي مع باقي المهام عندما تصبح في حالة تنفيذ (Executing)، أثناء ذلك يمكن للمهمة أن تخرج عن العمل لفترة زمنية محددةعُبِّر عنها بحالة التوقف المؤقت (Pended) وهذا يُكافئ في C/VxWorks تعليمة taskDelay(TimeOut). من الجدير ذكره أن المهمة تستطيع أن تقوم بعملية إيقاف مؤقت لنفسها فقط وليس للآخرين.




**الشكل (٤-١): مخطط أتومات المهمة ضمن نظام التشغيل VxWorks**

يمكن للمهمةأيضاً أثناء التنفيذ التعامل مع علبة البريد بإرسال الإشارتين msgQSend (التي تمثل وضع رسالة في علبة البريد وتحتاج إلى إشارة Ack للدلالة على نجاح عملية الإرسال) وmsgQReceive (التي تمثل قراءة رسالة من علبة البريد)، وبالتالي تحمل كل من هاتين الإشارتين في طياتهامتحولاً يدل على زمن التأخير المسموح به للرد المطلوب. يمكن أن يأخذ هذا المتحول إما قيمة عددية مباشرة أو قيمة NO_WAIT للدلالة على عدم الانتظار أو قيمة WAIT_FOREVER للدلالة على أن مرسل الإشارة سينتظر حتى تلبية طلبه.

- فإذا أرسلنا msgQSend(NO_WAIT) إلى علبة بريد ممتلئة يتم إهمال الرسالة والانتقال إلى الحالة ready مباشرة.
- أما إرسال msgQSend(WAIT_FOREVER) إلى علبة ممتلئة فيجعل المهمة تنتظر في حالة pended ريثما تتم القراءة من العلبة باستقبالها الإشارة msgQReceiveوإرسالها Ack للمهمة المنتظرة لتُأكد لها وصول الرسالة بنجاح. إن دخول المهمة حالة ready وعلبة البريد ممتلئة يسمح بتكرار استقبال الطلبيات وفق msgQSend إلى أن يصل عدد الرسائل الحد الأعظمي المسموح به (على عكس السيمافور الذي يستمر وفقه استقبال إشارة semGive حتى في حال تجاوز عدد طلبيات حجز السيمافور العدد الأعظمي الممكن)، وقد تم اختبار ذلك عملياً ضمن نظام Vxworks.

- إرسال (msgQSend(task_delay يؤدي إلى دخول المهمة وضعية انتظار مرتبط بزمن pended_Time وتبقى فيها إلى أن تتم القراءة من العلبة فإذا تمت عملية القراءة هذه قبل انتهاء الزمن تستقبل المهمة إشارة Ack، أما إذا تم تجاوز الزمن المسموح للاستجابة فإنه يتم استقبال إشارة error_msg.
- عند طلب المهمة القراءة msgQreceive من علبة بريد فارغة فإنها تنتظر في حالة waiting إلى أن تستقبل الإشارة msg_FirstWaiting التي تتضمن الرسالة المطلوبة.

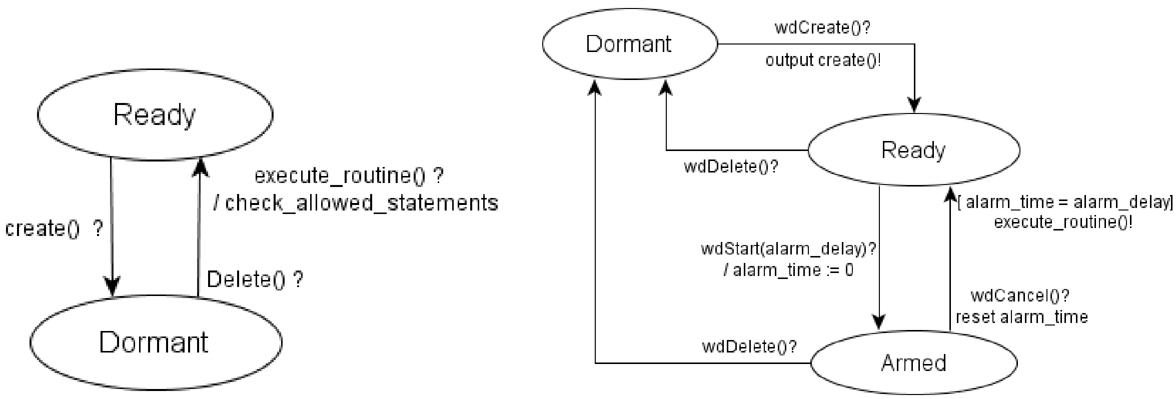
تتعامل المهمة مع السيمافور وفق إرسالها الإشارتين semTake (التي تمثل طلب لحجز السيمافور ويتم التأكد من أنه متاح باستقبال unit وفق الإشارة Ack_semTake) و semGive (تمثل تحرير السيمافور من قبل المهمة التي تحجزه حالياً)، وبالتالي تحمل هاتان الإشارتان في طياتها قيمة المتحول pid لتحديدمعر ّف للمهمة التي تطلب السيمافور والرد عليها حصراً، إضافة إلى متحول زمني يأخذ قيماً مماثلة للمتحول الزمني المرافق لإشارات علبة البريد المذكورة آنفاً تحر ّر المهمة السيمافور بإرسالها الإشارة semGive متابعةً عملها دون أي توقف. أما عند إرسالها semTake فإنها تنتظر الإشارة Ack_semTake لتتأكد من أن عملية الحجز قد تمت أو قد تستقبل الإشارة Ack_semError التي تدل على عدم إمكانية ذلك.

نلاحظ أن عمليتي الإرسال للأداتين mail Box و Semaphore تُعالجان بأسلوب مختلف بينما تتطابق الحالات الانتقالية للمهمة المرسلة في وضعية الاستقبال.

### 2.2.4 المؤقت (WatchDog (alarm:

يُنَفِّذ المؤقت مجموعة تعليمات في لحظة زمنية محددة تؤدي وظيفة معينة ضمن النظام. تمَّ مقابلة كل من المؤقت وبرنامج الخدمة المرفق به بعملية مستقلة لها حالاتها الخاصة، يصبح المؤقت (الشكل 4-٢) في حالة تلقيم armed لدى استقباله الإشارة (wdStart(alarm_delay وهنا يتم تفعيل عد ّاد زمني خاص به ويبقى المؤقت في هذه الحالة إلى أن تساوي قيمة العد ّاد قيمة المتحول alarm_delay (الدال على لحظة تنفيذ برنامج خدمة المؤقت بدءاً من لحظة تفعيل العد ّاد) عندها يتم إرسال إشارة execute_routine لبرنامج الخدمة لي ُنف ّذ تعليماته. يوجد عدد من التعليمات الخاصة التي يُسمح بمعالجتها في برنامج الخدمة وقد تم ّ مراعاتها برمجياً ضمن مترجمنا الهدف وهي التعليمات التي لا تؤدي إلى أي نقاط توقف ضمن النظام مثل التوابع التالية: ,msgQSend, semGive, wdSart, wdCancel, intUnlock intLock, taskResume, taskSuspend.

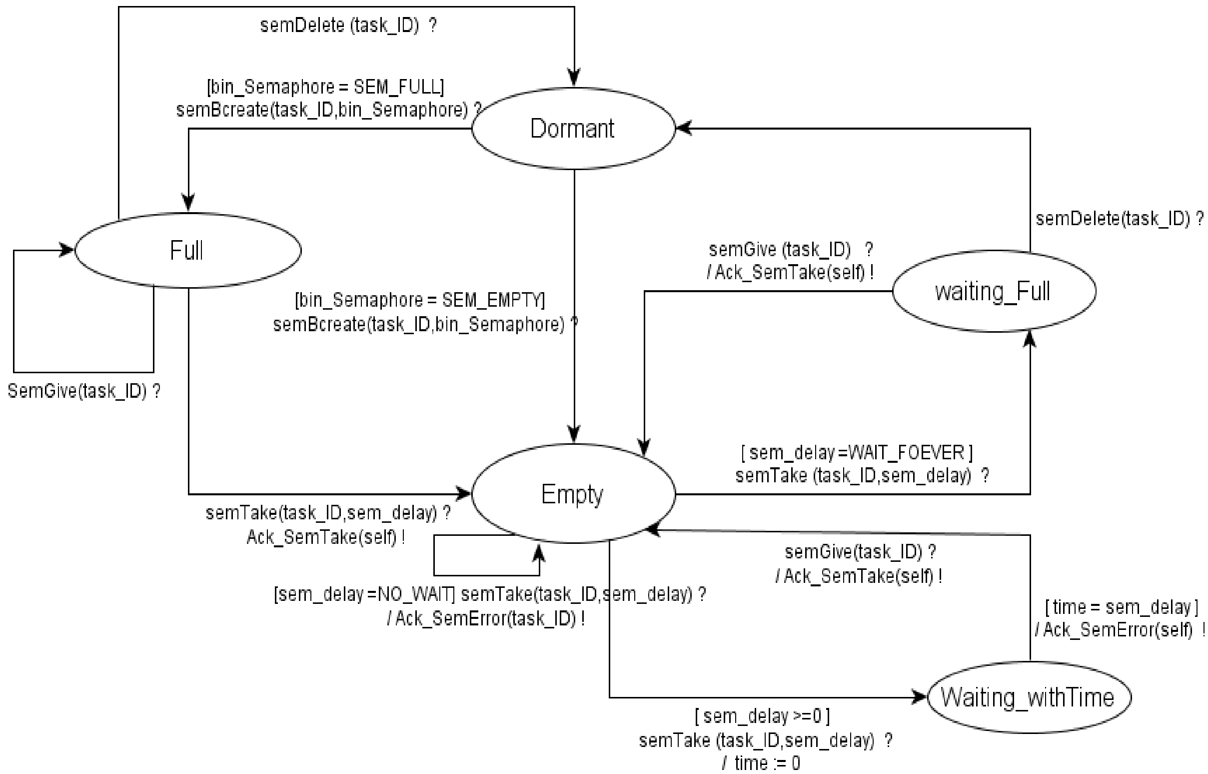
يمكن استقبال إشارة wdCancel في حالة armed فقط أي بعد أن يتم التلقيمحصراً ليتم تصفير قيمةالعدّاد الزمني دون أن يؤثر ذلك في العملية الخاصة ببرنامج الخدمة. يتم إنهاء المؤقت باستقبال wdDeleteوا رسال الإشارة Delete إلى برنامج الخدمة الذيُ نهي بذلك عمله.




**الشكل (4-٢): مخطط أتومات المؤقت وبرنامج الخدمة الموافق له ضمن نظام التشغيل VxWorks**

### 3.2.4 السيمافور الثنائي Binary Semaphore:

يتنوع السيمافور كما شرحناسابقاً وفق ثلاثة أنواع لكل منها أداؤه الخاص غير أن معظم التوابع المقابلة تكون ذات هدف برمجي مشترك ولضمان صحة عمليات إرسالو استقبال الإشارات المتبادلة بين السيمافورات التي قد توجد ضمن النظام تم التصريح عن نمط البيانات (pid) لتحديد العنصر المرسل للإشارة بدقة وليقابل المتحول المحدد للسيمافور المرافق لتابع الإنشاء. سنعرض أولاً السيمافور الثنائي الذي تمت مقابلته بعملية (الشكل 4-٣) تتألف من عدة حالات تبدأ لحظة بنائه بالحالة Empty أو Full وذلك يتحدد بقيمة المتحول bin_Semaphore المرافق للإشارة semBCreate، الذي تم التصريح عنه وفق النمط التعدادي Initial_state ليأخذ إحدى القيمتين SEM_EMPTY وSEM_FULL. يمكن أن يستقبل السيمافور الثنائي إشارة semGive عدة مرات دون أن تتغير حالته Full ويكفي استقباله semTake مرة واحدة فقط لينتقل إلى الحالة Empty كما أن إعادة استقباله الإشارة semGive لمرة تالية تعيده إلى وضعية Full.




**الشكل (٤-٣): مخطط أتومات السيمافور الثنائي ضمن نظام التشغيل VxWorks**

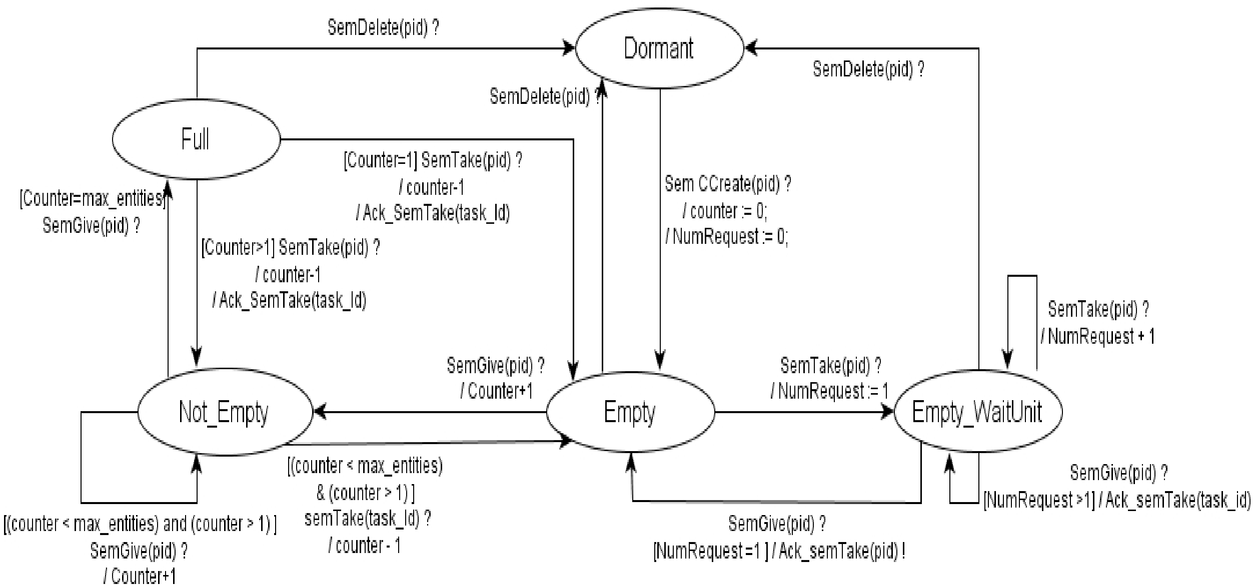
عندما يكون السيمافور في حالة Empty ويستقبل semTake المتضمنة للمتحول الزمني (الذي تم توضيحهسابقاً ) فإن انتقالاته تختلف حسب قيمة هذا المتحول وذلك كالتالي:

- يؤدي استقبال semTake(NO_WAIT) إلى البقاء في حالة Empty مع إرسال الإشارة Ack_semErrorالتي تدل على أن السيمافور غير متاح فلا يمكن القيام بالاستجابة المطلوبة لهذه الإشارة مباشرة.
- يؤدي استقبال semTake(WAIT_FOREVER) للانتقال إلى waiting_Full ويبقى في هذه الحالة فترة زمنية غير محددة إلى أن ترد الإشارة semGive فيرسل Ack_semTake وينتقل إلى حالة Empty.
- يؤدي استقبال semTake(sem_delay) إلى الحالة waiting_withTime وهنا قد يقوم السيمافور بإرسال الإشارة Ack_semTake أو الإشارة Ack_semError إلى المهمة الطالبة وذلك يرتبط بقيمة الزمن المتاح للرد وفيما إذا تم استقبال الإشارة semGive قبل تجاوز ذلك الزمن. نلاحظ أن إشارتي semGive وsemTake ليستا بحاجة إلى نقطة توقف زمني للمعالجة كما هو الحال في علبة البريد (التي سيتم شرحها لاحقاً ).

### 4.2.4 السيمافور التعدادي Counter Semaphore:

يعمل هذا السيمافور (الشكل 4-٤) بآلية مشابهه لآلية السيمافور الثنائي غير أنه يتميز بوجودعدّاد counter يدل على عدد مرات تحرير المصدر المطلوب أي تزداد قيمته باستقبال الإشارة semGive. يؤدي استقبال semCcreate إلى الانتقال من حالة dormant الابتدائية إلى الحالة Empty. التي يمكننا انطلاقاً منها تمييز ما يلي:

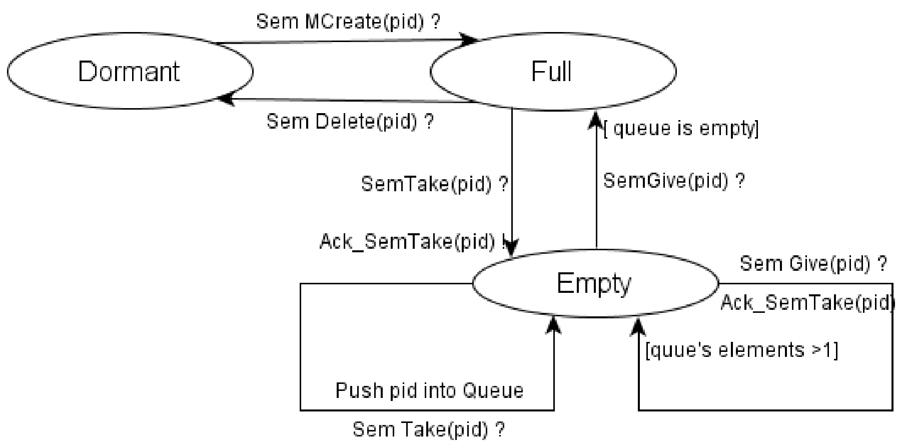
- استقبال السيمافور الفارغ للإشارة semTake ينقله إلى Empty_waitUnit ويؤدي إلى زيادة قيمة المتحول NumRequest الدال على عدد المهام الطالبة حجز السيمافور ويبقى في حالته هذه مهما تكرر استقبال الإشارة semTake الذي يترافق بزيادة قيمة المتحول السابق. أما استقبال الإشارة semGive انطلاقاً من Empty_waitUnit فيترافق بإرسال Ack_semTake إلى المهمة الطالبة والانتقال إلى الحالة Empty وذلك عندما يكون هناك مهمة واحدة فقط تطلب حجز السيمافور أي أن قيمة المتحول NumRequest تساوي الواحد أما في حال وجود أكثر من مهمة منتظرة فإن حالة السيمافور لا تتغير.
- استقبال السيمافور Empty للإشارة semGive يؤدي إلى الحالة Not_Empty ويبقى فيها مهما تكررت هذه الإشارة إلى أن يبلغ المتحول counter قيمة الحد الأعظمي المسموح المحددة بالمتحول max_entities (المرافق لإشارة semCcreate) فينتقل عندها إلى الحالة Full أما لدى استقباله semTake فإنه يرسل Ack_semTake (تتعدّل وضعية المهمة المرسلة لإشارة semGive بمجرد استقبالها Ack_semTake وتنتقل إلى الحالة Executing كما شرحنا سابقاً).




**الشكل (4-٤): مخطط أتومات السيمافور التعدادي ضمن نظام التشغيل VxWorks**

### 5.2.4 سيمافور الاستبعاد المتبادل Mutual Exclusion Semaphore:

قمنا بمقابلة هذا السيمافور (الشكل 4–٥) بعملية تترافق مع رتل انتظار يعمل وفق مبدأ FIFO يتم فيه تخزين معرّفات المهام المنتظرة. تتم عملية التخزين هذه عند استقبال السيمافور إشارة semTake وهو فارغ فهنا لا يمكن الرد على المهمة الطالبة لعملية الحجز مباشرة وإنما يُحتفظ بمعرّفها في نهاية الرتل إلى أن ترد الإشارة semGive فتُعالج هذه الطلبات المكدّسة اعتباراً من بداية الرتل الواحدة تلو الأخرى وذلك بإرسال Ack_semTake للمهمة ذات المعرّف الموجود في المقدمة ليتم تخديمها أولاً . ينتقل هذا السيمافور إلى الحالة Full لدى استقباله إشارة semMCreate وهو في حالة dormant.




**الشكل (4–٥): مخطط أتومات السيمافور الاستبعاد المتبادل ضمن نظام التشغيل VxWorks**

### 6.2.4 صندوق البريد (رتل الرسائل) MessageQueue:

تُعتبر الحالة dormant الحالة الابتدائية لعلبة البريد (الشكل 4–٦) وتعني أنها غير جاهزة ready لاستقبال الرسائل (غير مبنية)، وتبقى هكذا إلى أن تستقبل الإشارة msgQCreate فتنتقل إلى الحالة Empty. تترافق علبة البريد مع عدد من الإشارات.

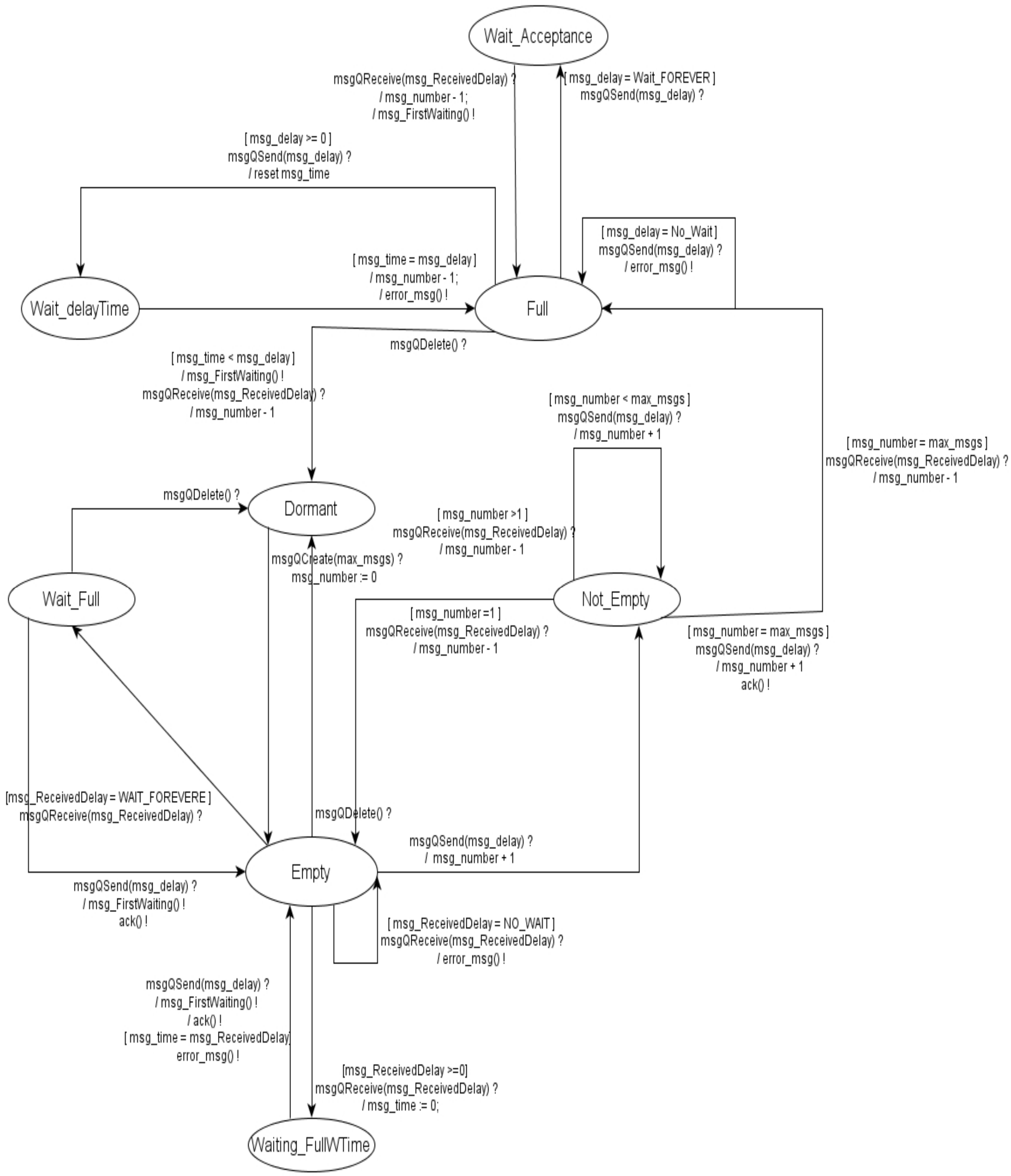
- يؤدي استقبال العلبة الفارغة لإشارة msgQSend إلى الانتقال للحالة Not_empty وتبقى في هذه الحالة طالما كان عدد الرسائل المستقبلة أصغر من الحد الأعظمي المسموح به فإن تساويا انتقلت إلى الحالة Full نميّز انطلاقاً من هذه الحالة الانتقالات التالية:
  - يؤدي استقبال msgQSend(WAIT_FOREVER) إلى wait_Acceptance وتبقى العلبة في هذه الحالة ريثما ترد الإشارة msgQReceive فيرسل Ack للمهمة المُرسِلة سابقاً لإشارة msgQSend للتأكيد على الاستقبال الصحيح لها.

- يؤدي استقبال msgQSend(msg_delay) إلى الانتقال للحالة wait_delayTime وهنا قد تتم عملية القراءة من العلبة باستقبالها msgQReceive قبل انتهاء الزمن المسموح أو قد ينتهي الزمن دون ذلك فترسل العلبة إشارة error_msg.
- يؤدي استقبال الإشارة msgQSend(NO_WAIT) والعلبة ممتلئة، إلى إرسال إشارة error_msg إلى المهملة ُرسِ لة دون أي تغيير لحالة علبة البريد.

- يتم استقبال الإشارة msgQReceive والعلبة فارغة، وفق معالجة مماثلة لحالة استقبال الإشارة msgQSend، حيث تترافق عملية القراءة هذه بمتحول زمني يأخذ إحدى القيم الثلاث التيدُ حدد التأخير الزمني المسموح للاستجابة المطلوبة.
- القراءة المتاحة من العلبة غير الفارغة (Not_empty) وفق استقبال إشارة msgQReceive يتعلق بعدد الرسائل المحتواة في العلبة.

- عدد الرسائل ضمن العلبة أكثر من رسالة واحدة وأقل من العدد الأعظمي المسموح به ضمن العلبة وفق هذه الحالة لا تتغير وضعية العلبة مهما تم استقبال الإشارة msgQReceiveوا ٕنما ينعكس ذلك على إنقاص عدد الرسائل الموجودة في علبة البريد فقط.
- عدد الرسائل يساوي لرسالة واحدة، تنتقل العلبة مباشرة للحالة Empty.
- عدد الرسائل يساوي العدد الأعظمي المسموح به ضمن العلبة، عندها تصبح العلبة غير فارغة (Not_empty).

هكذا نجد أن كلاً من الإشارتين msgQSend و msgQReceive تحتاج إلى نقطة توقف ريثما تتم المعالجة واستقبال الرد وفق الاستجابة المنتظرة.



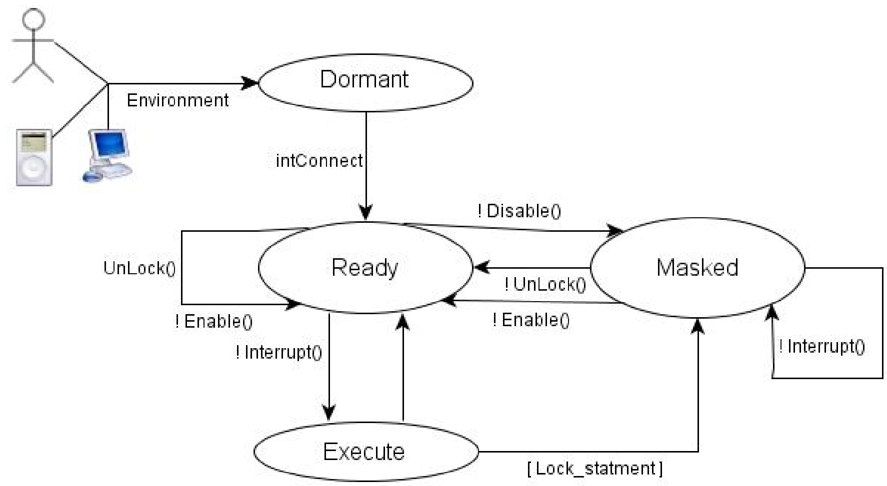

**الشكل (4-٦): مخطط أتومات علبة البريد ضمن نظام التشغيل VxWorks**

### 7.2.4 برنامج خدمة المقاطعة ISR (Interrupt Service Routine):

يتكوّن برنامج خدمة المقاطعة من مجموعة تعليمات برمجية تؤدي مهمة معينة ضمن النظام يرتبط تنفيذها بحدث ما ينجم عن الوسط الخارجي، بحيث لدى صدور هذا الحدث يتم إيقاف كافة المهام التي كانت

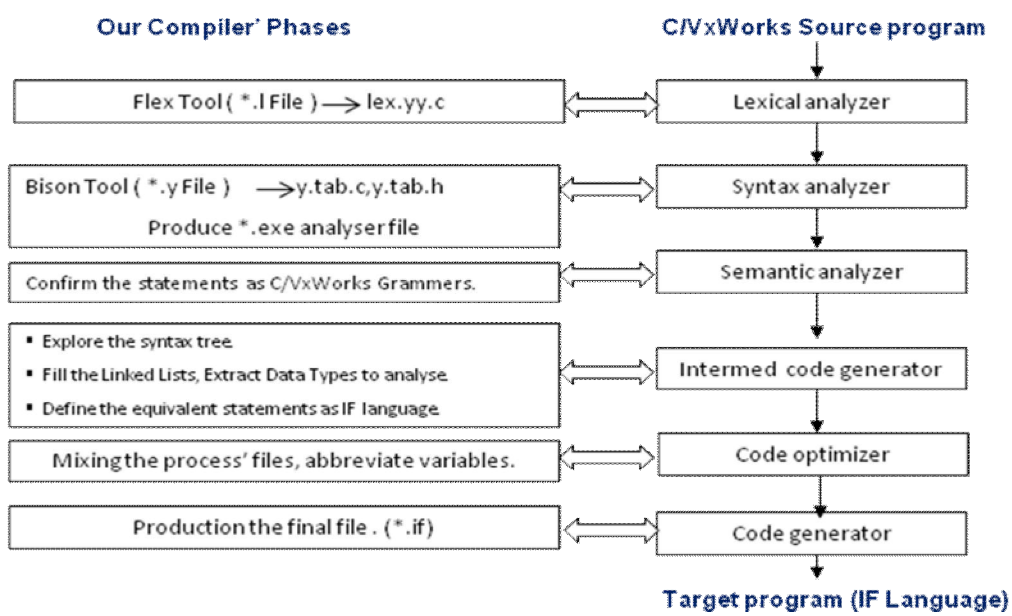
قيد العمل والانتقال لتنفيذ تعليمات برنامج الخدمة ومن ثم وبعد الانتهاء من تنفيذه يتم إعادة المهام المتوقفة إلى وضعية العمل لتتابع ما كانت عليه، إن أسلوب المعالجة هذا يقتضي عدم السماح ضمن برنامج خدمة المقاطعة بوجود أي تعليمة قد تؤدي إلى نقاط توقف، مثلاً لا يمكننا ضمنه كتابة التعليمة SemTake لأنه في حال كان السيمافورمحجوزاً من قبل مهمة أخرى ستؤدي هذه التعليمة إلى إيقاف العمل والانتظار ريثما يتم تحرير السيمافور المطلوب وهذا لا يتوافق مع عمل المقاطعة، كما أن عملية إنشاء غرض جديد ضمن برنامج خدمة المقاطعة غير مسموحة لأنها قد تؤدي إلى الانتظار في حال كانت الذاكرة ممتلئة.

قمنا بمقابلة برنامج خدمة المقاطعة بعملية مكّونة من عدة حالات (الشكل 4-٧) تبدأ بالحالة Dormant إلى أنْ يُنشأ برنامج خدمة المقاطعة وفق التابع intConnect الذي يقوم بربط شعاع المقاطعة (vector) مع هذا البرنامج (VOIDFUNCPTR routine) عندها يتم الانتقال إلى حالة الجاهزية (Ready) أي يصبح البرنامجقابلاً لاستقبال إشارة المقاطعة الخاصة به، وهنا يبقى البرنامج ضمن هذه الحالة مهما تستُقبلت الإشارة المؤدية إلى إلغاء القفل (Unlock) أو الإشارة (Enable) التي تقوم بتمكين المقاطعات المتوقفةسابقاً أما استقبال الإشارة Disable فيؤدي إلى الانتقال للحالة masked إلى أنتُستقبل إشارة (Unlock) أو إشارة (Enable). يتم تفعيل برنامج خدمة المقاطعة في حالة Execute فقط التي يتم الانتقال إليها باستقبال إشارة المقاطعة Interrupt من الوسط الخارجي Environment مما يؤدي إلى الانتقال إلى جسم البرنامج وترجمة تعليماته كاملة (نقوم بمقابلتها مباشرة بتعليمات لغة IF) ومن ثم بانتهاء هذه التعليمات تنتقل العملية المكافئة للمقاطعة إلى الحالة Ready ثانية، إلا إذاتضمن البرنامج تعليمة القفل المقابلة للتابع intLock الذي يردحصراً ضمن سياق برنامج خدمة المقاطعةمحدداً مدخل المقاطعة المستقبلة لعملية القفل، لم نقابل ضمن مترجمنا هذا التابع بأي إشارة لعدم وجود حاجة لذلكوا إنما لدى وجوده في برنامج الخدمةتُترجم كافة تعليمات البرنامج دفعة واحدة ثم في نهاية التنفيذ يتم القفل بنقل التنفيذ إلى حالة masked (علماً أنه لا يوجد أي خلاف في التنفيذ في حال وجد هذا التابع في برنامج الخدمة مع تعليمات سابقة أو لاحقة له لأن تعليمات إرسال الإشارات واستقبالها ضمن النظام تأخذ زمناً للتنفيذ قدره صفر).




**الشكل (4-٧): مخطط الأتومات لبرنامج خدمة المقاطعة ضمن نظام التشغيل VxWorks**

بعد التوصل الى المقابلة بين لغتي IF و C/VxWorks وبرمجة الأتومات الخاص بكل عنصر ومقابلته بتعليمات لغة IF أصبح بإمكاننا بناء المترجم المنشود وتوليد الشيفرة النهائية له (الشكل 4-8).




**الشكل (4-8): مقابلة المراحل العامة للمترجم مع خطوات بناء مترجمنا**

## 3.4 القواعد المتبعة في عملية المقابلة ومعالجة المشاكل البرمجية:

لقد قمنا بدراسة تفصيلية لقواعد لغة C/VxWorks ولغة IF وذلك لإيجاد نقاط التشابه والاختلاف بين هاتين اللغتين وفق جميع التعليمات الممكنة سواء فيما يتعلق بأنماط البيانات المتاحة للتصريح عن المتحولات وكيفية تسميتها والتعامل معها أو فيما يختص بأسلوب توصيف سلوكية النظام البرمجي المدروسإضافةً إلى قضايا أساسية أخرى كانلابدّ لنا من أخذها بعين الاعتبار حتى تتم عملية المقابلة بالشكل الصحيح. سنذكر حالياً أهم هذه القضايا البرمجية التي صادفتنا خلال عملية المقابلة وكيفية معالجتها.

### 1.3.4 قواعد تسمية المتحولات وأنواع بياناتها:

تتقارب قواعد التسمية بين لغتي IF و C/VxWorks فكلتا اللغتين حساسة لحالة الأحرف (case sensitive) وتُميّز بين وضعية الأحرف الكبيرة والصغيرة أثناء استخدامها لكلمات اللغة المحجوزةتبيّن لنا أثناء مقابلتنا لأنواع بيانات اللغتين (الجدول 4-1) أن هناكتقابلاً بين أنواع محددةو أنواع أخرى لا مثيل لها في اللغة المناظرة، على سبيل المثال نمط البيانات clock و self في لغة IF ليس له أي نظير في C/VxWorks وأنماط البيانات char و float المستخدمة في C/VxWorks لا يمكن مقابلتها بأنماط لغة IF.

| الأنماط ضمن لغة IF | الأنماط في C/VxWorks |
|---|---|
| boolean (true \| false) | short |
| ***integer*** | ***int*** |
| float | long |
| self | float |
| nil | double |
| pid | void |
| Abstract data type | signed/ unsigned |

**الجدول(4-1): أنماط البيانات ضمن اللغتين**

### 2.3.4 مقابلة التعليمات البرمجية الأساسية للغة C ضمن نظام VxWorks:

يوضح الجدول (٤-٢) التعليمات المتقابلة بين لغة C/VxWorks ولغة IF. علماً أن معظم هذه التعليمات تحتاج إلى تغييرات نصية للكلمات غير المفتاحية قبل أن تصبح جاهزة للتطبيق المباشر ضمن البيئة البرمجية الهدف وذلك لأننا اعتمدنا في هذا الجدول على توصيف القواعد وفق BNF (الكلمات المفتاحية للغة مميزة بخط غامق). يدل N/A على عدم وجود مقابل للتعليمة البرمجية وفق اللغة الأخرى.

| Statements | C - Statements | IF – Intermediate Formula Language |
|---|---|---|
| **Variable Declaration** | Simple-type variable-name; | Var-decl:= **Var** var-id type-id { **private** \| **public** };<br>Fpar-decl::= [ **in** \| **inout** \| **out** ] fpar-id type-id |
| **Timer Declaration** | N/A | **Var** variable-name **clock**; |
| **Constant Declaration** | **Const** Variable-name= value ;<br>**#define** Variable-name value | Const-decl ::= Const const-id [ = const] ;<br>Const ::= **true** \| **false** \| **integer** \| **float** \| **self** \| **nil** |
| **Variable Assignment** | Variable-name = value; | **Task** expression := expression; |
| **Set Time_ variable** | N/A | **set** expression := expression ; |
| **Reset Time_variable** | N/A | **reset** expression ; |

| Statements | C - Statements | IF – Intermediate Formula Language |
|---|---|---|
| **Enum Declaration** | **Enum**<br>{ Variable-name1,Variable-name2 , etc }<br>enum-name; | **Type** type-id = type ;<br>Type ::= **enum** const-id { , const-id }* **endenum;** |
| **Array Declaration** | Simple-type variable-name [size]; | **array** [ const .. const ] of type-id; |
| **struct Declaration** | **struct** Variable-name<br>{ Variable Declaration; etc }; | **record** {field-decl }* **endrecord** ;<br>Field-decl ::= Field-id type-id; |
| **Casting Statement** | (simple-type) variable-name; | cast-op ::= {type-id \| process-id \| signalroute-id } |
| **Condition Statement** | **If** (condition)<br>{statements}<br>**else**<br>{statements} | **If** (condition) **then**<br>{statements}<br>**else**<br>{statements}<br>**endif** |
| **Iteration Statement** | **while** (condition)<br>{statements} | **while** (condition) **do**<br>{statements}<br>**endwhile** |
| **Skip Statement** | **Break**; | **Skip**; |
| **Continue Statement** | **Continue**; | N/A |
| **Function Declaration** | Return-type func-name (parameters)<br>{<br>/* C/VxWorks Statements */<br>}; | Type-id function-id ( [ type-id { , type-id }*]) ;<br>/* The body of function as Procedure-declaration */ |
| **Procedure eclaration** | N/A | **Procedure** procedure-id ;<br>[fpar fpar-decl {, fpar}*;]<br>[returns type-id ; ]<br>[{# c code # }]<br>**Endprocedure** ; |
| **Return Statement** | **return** (**variable_name** \| **value**); | **returns** type-id ; |
| **Call Function** | Variable-namem = funcname (passed-parameters); | [ expression :=] **call** procedure-id ([ expression { , expression }*]); |
| **Recursive procedures** | Call the same function into it's body | [ expression :=] **fork** process-id([ expression { , expression }*]);<br>[ expression :=] **fork** signalroute-id([ expression { , expression }*]); |
| **Input Statement** | simple-type Variable-name = **getchar** (Parameters); | N/A |
| **Print Statement** | **printf**(list of variables); | **informal** "string information" ; |
| **Especially Statements** | N/A | **kill** expression ;<br>**stop**;<br>**abstract** { function-decl }* **endabstract**<br>**range** const .. const<br>**string** [ const ] **of** type-id<br>**provided** expression;<br>**when** constraint; |
| **Mutually exclusive access**<br>physical or logical resource | N/A | **resource** buffer_name;<br>**Acquire** buffer_name;<br>**Release** buffer_name; |

| Statements | C - Statements | IF – Intermediate Formula Language |
|---|---|---|
| **Selection Statement** | **repeat , switch , case** | N/A |
| **Logical operators** | ! \| \|\| \| && | **Not \| or \| and** |
| **Compare operators** | < \| > \| = \| >= \| <= \| == | < \| > \| = \| >= \| <= \| = |
| **Comments** Statements at same line | // comments | /* comments */ |
| **Comments :** Multi Lines | /* comments */ | /* * comments */ |

**الجدول(٤-2): التعليمات الأساسية المتقابلة بين اللغتين**

### 3.3.4 مشاكل في البنية الهيكلية القواعدية للغة IF:

تنشأ التعارضات الهيكلية بين اللغتين نتيجة لاختلاف قواعد لغة IF عن باقي لغات البرمجة الأخرى حيث تعتمد لغة IF تعليمات خاصة لتوصيف برامجها المكوّنة من نظام يحتوي عدة عمليات ذات حالات وإشارات مختلفة. إن التميُّز القواعدي لتعليمات لغة IF عن تعليمات لغة C/VxWorks فرض علينا جهداً إضافياً من خلال إجراء اختبارات خاصة للتأكد من صحة تعليمات المقابلة المقترحة. صادفنا خلال عملية المقابلة هذه العديد من المشاكل البرمجية التي تطلبت قراراً برمجياً قطعياً لتحديد أسلوب المعالجة الصحيح الذي يضمن حصولنا على برامج تحترم قواعد لغة الخرج بشكل دقيق وتؤدي السلوكية المطلوبة منها. سنوضح حالياً بعض هذه المشاكل الناجمة عن تميّز تعليمات لغة IF (على سبيل الذكر لا الحصر).

- عدم إمكانية مقابلة التعليمة الشرطية if (condition) {…} else {…} بتعليمة شرطية مماثلة مباشرة. رغم أن هذه التعليمة موجودة ضمن لغة IF وتؤدي الغرض البرمجي نفسه إلا أن لغة IF تُميّز في تعليماتها الشرطية بين أنماط البيانات المختبرة فيما إذا كانت تتضمن دلالة زمنية أم لا. فلاختبار التعليمات الشرطية ذات القيد الزمني تُستخدم تعليمة when ولاختبار التعليمات الشرطية ذات القيد غير الزمني تُستخدم تعليمة provided أما تعليمة if الشرطية فلا يمكن ورودها في تعليمات البرنامج إلا ضمن transition كما يُمنع كتابة تعليمة (nextstate …;) ضمن كتلة تعليماتها (من التعليمات المسموحة ضمنها set (reset, output, لذا كان لابدّ لنا من تمييز هذه الحالات برمجياً .
- لايمكننا استخدام تعليمة provided لاختبار قيمة متحول مرافق لإشارة مستقبلة مباشرة ضمن الحالة المتضمنة لها (في حال كانت هذه التعليمة هي الوحيدة المناسبة للاختبار المطلوب). لأن هذه التعليمة تسمح فقط باختبار قيم متحولات مستقبلة خلال حالات سابقة. لمعالجة ذلك نلجأ إلى التصريح عن حالة انتقالية وسيطة يتم توصيفها على أنها غير مستقرة (#unstable) وعند انتقال التنفيذ إليها نقوم بالاختبار المطلوب. هذا ما يفسر تعريفنا للحالة decision_1 غير المستقرة ضمن السيمافور الثنائي

لتكون الحالة التالية لحالته الابتدائية وليتم فيها اختبار قيمة المتحول bin_Semaphore المحدد لوضعية السيمافور لحظة تهيئته والذي تم معالجته وفق الشيفرة البرمجية التالية:

```
state dormant #start ;
    input semBcreate(task_ID,bin_Semaphore);
    nextstate decision_1;
endstate;
state decision_1 #unstable ;
    provided (bin_Semaphore = SEM_FULL);
    nextstate full;
    provided (bin_Semaphore = SEM_EMPTY);
    nextstate empty;
endstate;
```

كما أن وجود أكثر من تعليمة provided ضمن الحالة الواحدة يتطلب أن تُتبع كل تعليمة منها بتعليمة nextstate لتحديد الحالة الانتقالية التالية الناجمة عن تحقق الشرط المرافق وفي حال عدم مراعاة ذلك يتوقف التنفيذ وتظهر لنا عبارة تدل على وجود خطأ قواعدي (Syntax Error) في البرنامج المترجم.

- إن الدقة في قواعد لغة IF فرضت علينا جهوداً برمجية إضافية، على سبيل المثال الحالة الابتدائية لكل عملية يجب أن يُصرّح عنها بالتعليمة (#start ;) وهنا لابدّ من الانتباه إلى ضرورة وجود فراغ قبل الفاصلة المنقوطة، علماً أنه في مواضع أخرى يؤدي وجودها إلى خطأ قواعدي. كما أنه لا يوجد قسم خاص للتصريح عن المتحولات المحلية الخاصة بعملية معينة وإنما يمكننا التصريح عن المتحولات المعالجة ضمن أي حالة مباشرة في أي جزء من جسم العملية. كما تمتاز لغة IF بتنوّع أنماطها حيث يتم اعتبار الاسم المصرّح وفقه عن أي عملية أو حالة في النظام على أنه نمط بيانات جديد يمكن التعامل معه ضمن تعليمات البرنامج إضافة إلى إمكانية إجراء عملية تحويل قسري للمتحولات ذات أنماط أخرى إليه وهذا غير متاح ضمن لغات البرمجة الأخرى.
- تم صياغة قوالب ثابتة لمعالجة إشارات خاصة ضمن المهمة بالشكل الذي يضمن لنا معالجة كافة الإشارات المتبادلة في النظام. على سبيل المثال لدى معالجة إشارتي taskSuspend,taskResume اللتين يمكن للمهمة استقبالهما حصراً في إحدى الحالتين waiting أو ready. لقد قمنا باعتبار استقبال المهمة الجاهزة لإشارة taskSuspend يؤدي إلى انتقالها إلى الحالة pended ريثما يتم استقبال الإشارة taskResume لتعاود المهمة وفقها الانتقال إلى حالة الجاهزية السابقة ذاتها إذ لابدّ ضمن لغة C/VxWorks أن تُتبع إشارة الإيقاف المؤقت بإشارة استمرارية لمتابعة العملية مهامها المطلوبة أما في حال لم يتم استقبال إشارة taskSuspend فإن المهمة تنتقل مباشرة من وضعية الجاهزية إلى التنفيذ دون أن يؤثر وجود الإشارات الأخرى في مخطط الحالة النهائي. كما أن عدم وجود الإشارة ضمن الشيفرة البرمجية للبرنامج المطلوب ترجمته لا ينعكس على مخطط الحالات الشامل وإنما يأخذ عندها النظام

المسار الاحتمالي الآخر لتبادل الإشارات بين حالات النظام وقد قمنا بصياغة التعليمات المقابلة لذلك وفق التالي:

```
state ready_0;
   input taskSuspend();
   nextstate pended_0;
   nextstate executing_0;
endstate;
state pended_0;
   input taskResume();
   nextstate ready_0;
endstate;
state pended_0;
   input taskResume();
   nextstate ready_0;
endstate;
```

أما استقبال إشارة التوقف المؤقت taskSuspend والمهمة في حالة انتظار تحرير السيمافور فإن ذلك سيؤدي إلى انتقالها إلى حالة pended وفي هذه الحالة لابدّ من استقبال إشارتي Ack_SemTake و taskResume لتُعاود المهمة جاهزيتها وفي حال تم استقبال إشارة Ack_SemTake فقط فإن المهمة تحافظ على هذه الإشارة وتبقى في الحالة ذاتها ريثما تُستقبل إشارة taskResume. كما يتم اتخاذ الإجراء نفسه في حال تُقبلت الإشارة taskResume قبل إشارة التأكيد على كون السيمافور متاحاً للمهمة الطالبة، وقد تم تمييز ذلك من خلال المتحول Ack_signal الذي يأخذ القيمة (1) لدى استقبال الإشارة Ack_SemTake. التعليمات البرمجية المقابلة لهذه المعالجة وفق لغة IF هي:

```
state waiting_0;
   input taskSuspend();
      task Ack_signal := 0;
   nextstate pended_1;
   input Ack_SemTake(task_ID);
   nextstate ready_1;
   input Ack_SemError(task_ID);
   nextstate ready_1;
endstate;
state pended_1;
   input Ack_SemTake(task_ID);
      task Ack_signal:= 1;
   nextstate pended_1;
   input taskResume();
   nextstate desicion_0;
endstate;
state desicion_0;
   provided (Ack_signal = 0);
   nextstate waiting_0;
   provided (Ack_signal = 1);
   nextstate ready_1;
endstate;
```

- لم تُقابل الحلقة التكرارية اللانهائية {...}(;;)for المتاحة ضمن لغة C/VxWorks، التي تُكرر تنفيذ تعليماتها بشكل غيرمنتهٍ بتعليمة for المكافئة لها ضمن لغة IF. لأن مقابلتها بهذا الأسلوب يفرض

علينا التحكم بتسلسل الانتقال بين حالات النظام المقابل لبرامج لغة IF أثناء تنفيذها. إلا أننا فعلياً لا يُمكننا ذلكوإنما نتحكم بالإشارات المتبادلة ضمن النظام بذلك وتتحدد سياسة تنفيذ برامج لغة IF، إما بسياسة الإبعاد أي المهمة ذات الأولوية الأعلى هي التي تمتلك زمام التنفيذ، أو يمكن أن يتم التنفيذ حتى النهاية أي تتابع المهمة قيد العمل تنفيذ عملهاالمُسند لها دون أي مقاطعة من المهام الأخرى حتى انتهاء عملها. لذلك يتم وفق مترجمنا ترجمة تعليمات لغة IF ومقابلتها بما يوافقها من حالات ويتوقف تكرار هذه الحالات على سير التنفيذ.

- الاختلاف والتباين الكبير بين أنماط البيانات المتاحة من قبل اللغتين جعل عملية التصريح عن متحولات بأنماط متكافئةأمراً شبه مستحيل فلا يمكننا التعامل مع ثابت نصي رغم أن كلتا اللغتين تدعم تعريف الثوابت كما لا يمكننا القيام بتعريف مصفوفة نصية لعدم دعم لغة IF الأنماط (char,string). مما جعلنا نلجأدوماً إلى التصريح عن المتحولات الرقمية ذات المدلول الخاص حسب الحاجة (تم إيضاحهاسابقاً أثناء مقابلة التعليمات).
- نظراً لعدم وجود أي تعليمة ضمن لغة IF تدعم القيم التفاعلية المدخلة من قبل المستخدم فإنه لم يكن باستطاعتنا مقابلة تعليمة الدخل getchar بشكل مباشروإنما تم مكافئة هذه التعليمة وفق لغة IF بعملية مكوّنة من حالتين creating و working وفق الشكل التالي:

```
process getchar(1);
var time clock;
state creating #start ;
  set time := 0 ;
  nextstate working;
endstate;
state working;
   when time = 1;
   reset time;
   stop;
endstate;
endprocess;
```

إضافة لما سبق قمنا بعدد من المعالجات البرمجية الخاصة التي ساعدت بمجملها على إنتاج المترجم النهائي القادر على ترجمة تعليمات لغة الدخل بشكل صحيح وفق المطلوب.

## 4.4 اختبار تكاملية المترجم:

سنتحدث الآن عن الخطوة الأخيرة في بناء المترجم فبعد تحديد التقابلات بين تعليمات اللغتين وتوليد الشيفرة البرمجية النهائيةلابدّ من التأكد من تكاملية عمل مترجمنا والتحقق من صحة نتائجه. إن تقييم أداء أي برمجية يتم من خلال مقارنتها مع أدوات أخرى مماثلة تؤدي العمل المطلوب نفسه، لذلك فلتقييم مترجمنا لابدّ من دراسة متر جمات أخرى خاصة بنظم الزمن الحقيقي ومقارنة نتائج هذه المترجمات مع النتائج التي يتم

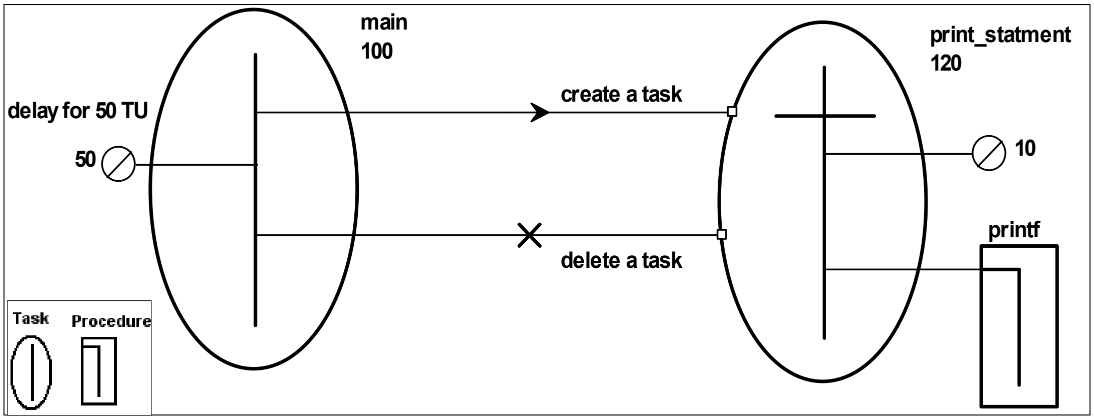
الحصول عليها من خلال مترجمنا. يمكن أن تشمل المقارنة توضيطاً لنقاط التشابه والاختلاف بين عمل المترجمات المعتمدة أو أن تتم دراسة عدد من المعايير التي تفيد في تحديد فعالية أداء كل منها كدراسة زمن التنفيذ ودرجة الدقة في تحديد الأخطاء المرتكبة في البرنامج المحلل وأخيراً تحديد مدى صحة نتائج الخرج كتابع لمعطيات الدخل. غير أننا عملياً لم نتمكن من العثور على أي مترجمات تعالج أنظمة الزمن الحقيقي بهذا الأسلوب لذا سنعتمد في اختبار مترجمنا على تحليل تعليمات برنامج لغة الدخل C/VxWorksو التحديد المسبق لما ينبغي أن يقابلها وفق لغة الخرج IF ومن ثم التحقق من ذلك وفق التطبيق العملي للبرنامج المترجم وفق مترجمنا. سنعرض الآن وبشكل مختصر مثالاً عن عملية التحويل التي يقوم بها مترجمنا لاختبار صحة الشيفرة البرمجية المقابلة لتعليمات لغة C/VxWorksوالمولّدة وفق لغة IF.

التطبيق التالي هو برنامج مكتوب بلغة C/VxWorks (ملف الدخل لتطبيقنا الشكل 4-9) ويحتوي على مهمتين الأولى هي Main والثانية Print_statement. تقوم المهمة Main في بادئ الأمر بخلق المهمة المسماة Print_statement وذلك باستخدام التعليمة taskSpawn ومن ثم تؤخّر نفسها لمدة ٥٠ وحدة زمنية باستخدام التعليمة taskDelay(50) وعند انتهاء زمن التأخير هذا تتابع عملها فتقوم بهدم المهمة الثانية باستخدام التعليمة taskDelete. وبشكلموازٍ multitask تعمل المهمة الثانية التي تتألف من حلقة لانهائية for(;;) يتم بداخلها التأخير لمدة قدرها ١٠ وحدات زمنية ومن ثم تتم عملية الطباعة باستخدام تعليمة printf.

```
/* headers declaration */
#include<tasklib.h>
#include<time.h>
#include<stdio.h>
#include<math.h>
/*variables decalaration */
local int functionid;
void print_statement(void);
int main(void)
{         functionid=taskSpawn("task1",20,0,200, print_statement,0,0,0,0,0,0,0,0,0,0);
          taskDelay(50);
          taskDelete(functionid);
          return 0;
}
void print_statement (void)
{     for(;;)
          {
               taskDelay(10);
               printf(" This statement is printed from the second print_statement function ");
          }
 }
```

**الشكل (4-9): البرنامج وفق لغة C/VxWorks**

تم توضيح المخطط العام لبرنامج الدخل وفق الشكل (4-10)، المرسوم باستخدام لغة LACATRE.




**الشكل (4-10): مخطط برنامج الدخل باستخدام Lacatre**

أثناء تحليلنا لهذا البرنامج وتحديد التعليمات التي يجب أن تقابله نجد أنه يحتوي فقط على مهمة واحدة task1 تم إنشاؤها وتفعيلها وفق التابع taskSpawn الذي يُعيد المعرّف functionid المقابل للمهمة المنشأة، كما نلاحظ وجود التابع الرئيسي main الذي يجب أن تتم مقابلته أيضاً بمهمة مستقلة وبذلك سيكون الخرج المتوقع هو عبارة عن نظام يتضمن عمليتين حيث أن كل مهمة وفق لغة IF تُقابل بعملية (process) واحدة فقط، أما التابع taskSpawn فيتم مقابلته بإشارة دخل وخرج (كما شرحنا سابقاً) لم ننتهِ من عملية المقابلة فمازال هناك عدد من التوابع البرمجية الأخرى التي تقود العمل وهي التابع (taskDelete) الذي سيتم مقابلته بإشارة خاصة لها الاسم نفسه تحمل بين طياتها المتحولات الضرورية لعمله إن وجدت. أما التابع (taskDelay) فيُقابل انتقال المهمة المتضمنة له إلى حالة الإيقاف المؤقت التي تُرفق بمتحول زمني خاص يضبط زمن استمرار هذه الحالة لتُعاود المهمة عملها لحظة انتهاء الزمن المحدد. سنشرح الآن مراحل تحويل هذا البرنامج وفق مترجمنا.

كما أشرنا سابقاً في الفصل الثالث، فلقد تم كتابة المحلل اللفظي والتركيبي باستخدام تقنيتي Lex وYacc. قمنا ضمن ملف الـ Lex بكتابة جميع التعابير النظامية التي يمكن أن ترد في ملف الدخل حيث يحتوي هذا الملف (155) تعبيراً نظامياً أما ملف الـ Yacc فيحتوي على (280) قاعدة وهي تقابل عدد القواعد الممكنة للغة. على سبيل المثال أثناء تحليل برنامجنا وفق الـ Lex وYacc فإن السطر التالي من الشيفرة أعلاه.

```
functionid=taskSpawn("task1",20,0,200,print_statement,0,0,0,0,0,0,0,0,0,0);
```

تقابله التعابير النظامية التالية:

functionid ←
التعبير النظامي المكافئ: *[0..9]*[a..z][a..z] ممكن أن يولّد هذا التعبير معرّفاً ما Id يبدأ حكماً بحرف يتبعه تسلسل ما من الأحرف أو الأرقام. يدل الرمز (*) على احتمال محرف أو أكثر أو عدم وروده إطلاقاً.

←
تعبير نظامي عبارة عن Id (كلمة مفتاحية للغة) يمتلك نفس صيغة التعبير السابق.

←
يقابل هذه الكلمة تعبير نظامي تولّد كلمة واحدة مكوّنة من عدد ما من المحارف (كلمة غير مفتاحية).

←
تعبير نظامي يولّد كلمة واحدة فقط (كلمة غير مفتاحية).

taskSpawn

task1

print_statement

200

تتم عملية مكافئة التعابير النظامية ضمن مرحلة التحليل اللفظي ثم تليها مرحلة التحليل القواعدي (القاعدة المقابلة لتعليمة taskSpawn التي تؤدي إلى خلق وتفعيل مهمةقد حُددت في الفصل الثالث ضمن الفقرة 1.4.3 قواعد BNF للغة C/VxWorks). أما في مرحلة تحليل المعاني المقابل لهذه التعليمة فيتم مثلاً التحقق من وجود تابع يمتلك الاسم (print_statement) ليتم الانتقال إليه أثناء تفعيل المهمة task1. أخيراً يتم توليد الشيفرة البرمجية النهائيةالمقابلة لهذا البرنامج كاملةً حيثُ قابل تعليمة taskSpawn بالتعليمات التالية:

- إشارة عامة ضمن النظام: input taskSpawn();
- إشارة خرج ضمن العملية الرئيسية: output taskSpawn() to ({print-statement}0);
- إشارة مستقبلة ضمن الحالة الابتدائية للعملية print-statement :

state dormant #start ; input taskSpawn(); nextstate ready-0;

بالتالي حصلنا على ملف الخرج النهائي (الشكل 4-11)، الذي هو عبارة عن برنامجٍ مثّل نظام يُدعى example_Task يتم ضمنه تبادل الإشارتين taskSpawn وtaskDelete بين العمليتين: العملية Main والعملية Print_statement. تمر العملية Main بعدد من الحالات الانتقالية تبدأ بالحالة dormant ومنها إلى الحالة ready فالحالة executing. تباشر العملية Main مهامهااعتباراً من هذه الحالة فتقوم بإنشاء العملية الثانية وتفعيلها وذلك بإرسالها الإشارة taskSpawn. ثم تقوم بإدخال نفسها في حالة إيقاف مؤقت pended يترافق مععدّاد زمني task_time وهو متحول من النمط clock. يسمح هذا الإيقاف للعملية الأخرى Print_statement باستلام زمام العمل والقيام بما هو مطلوب منها إلى أن يبلغ المتحول السابق القيمة (50) عندها تنتقل العملية Main إلى وضعية الجاهزية ثانية ومنها إلى الحالة executingمرسلةً الإشارة taskDelete لتهدم العملية الثانية ومن ثم يتوقف عملها وفق التعليمة stop. أما العملية Print_statement فتمر بعدة حالات تبدأ بالحالة dormant التي يتم وفقها استقبال الإشارة taskSpawn المرسلة من قبل العملية Main والانتقال إلى الحالة ready ومنها إلى الحالة executing. بعد ذلك تتوقفمؤقتاً pended مع تفعيلعدّاد زمني مرافق لها وتبقى هكذا إلى أن يبلغ زمن التأخير القيمة

(10) عندها تنتقل إلى الحالة ready ثانية لتقوم بطباعة العبارة المطلوبة وفق تعليمة informal ثم تستقبل الإشارة taskDeleteلتُهدم وتنتقل إلى الحالة الابتدائية dormant. يمكن توضيح الحالات المقابلة لملف الخرج هذا وفق مخطط مماثل لأتومات المهمة الموضح في الشكل (4-١).

```
system onetask;
    signal taskSpawn();
    signal taskSuspend();
    signal taskResume();
/* headers decalaration */ /* variables decalaration */
    signal taskDelete();
process functionid(1) ;
    var task_delay integer; var task_time  clock; /* Variables Declaration */
state dormant #start ;
   input taskSpawn();
   nextstate ready_0;
endstate ;
state ready_0;
   input taskSuspend();
   nextstate pended_0;
   nextstate executing_0;
endstate;
state pended_0;
   input taskResume();
   nextstate ready_0;
endstate;
state executing_0;
   set task_time := 0;
   nextstate pended_1;
endstate;
state pended_1;
   when task_time = 10;
   nextstate ready_1;
endstate;
state ready_1;
   input taskSuspend();
   nextstate pended_2;
   nextstate executing_1;
endstate;
state pended_2;
  input taskResume();
   nextstate ready_1;
endstate;
state executing_1;
   informal "This statement is printed from the second print_statement function ";
   nextstate ready_2;
endstate;
state ready_2;
   nextstate dormant;
endstate;
endprocess;
process main(1);
   var task_delay integer; var task_time  clock;  /* Variables Declaration */
state dormant #start ;
   nextstate ready_0;
endstate ;
```

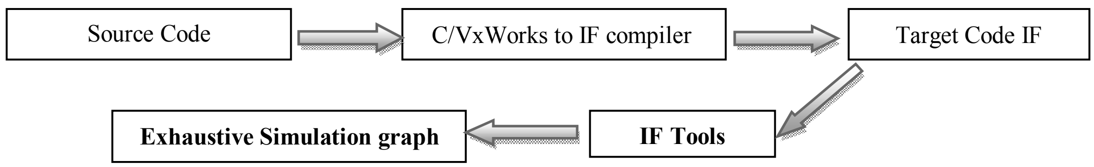
```
state ready_0;
   nextstate executing_0;
endstate;
state executing_0;
   output taskSpawn() to ({functionid}0);
   set task_time := 0;
   nextstate pended_0;
endstate;
state pended_0;
   when task_time = 50;
   nextstate ready_1;
endstate;
state ready_1;
   input taskSuspend();
   nextstate pended_1;
   nextstate executing_1;
endstate;
state pended_1;
   input taskResume();
   nextstate ready_1;
endstate;
state executing_1;
   output taskDelete() to ({functionid}0);
   nextstate ready_2;
endstate;
state ready_2;
   stop;
endstate;
endprocess;
endsystem;
```

**الشكل (4-11): البرنامج المقابل وفق لغة IF**

تم التصريح عن المتحولات ضمن كل عملية بصورة مستقلة عن الأخرى لأن لكل منها متحولاتها الخاصة، وبما أنه يُمنع وجود حالتين لهما التسمية ذاتها ضمن مخطط الحالة الخاص بالعملية الواحدة فقد تمّ برمجياً إلحاق قيم رقمية متسلسلة مُرفقة بأسماء الحالات لتمييزها فيما بينها حيث قد توجد الحالة التي تُعبّر عن الطبيعة التنفيذية نفسها أكثر من مرة ولكن برقم تسلسلي وتعليمات مختلفة وهذا ضروري لدقة العمل. على سبيل المثال نلاحظ ضمن العملية main وجود حالتي تنفيذ (executing_0, executing_1) مستقلتين تؤدي المهمة في حالة التنفيذ الأولى عدة تعليمات ثم تدخل وضعية تأخير زمني وبعد معاودتها العمل تُتابع ضمن الحالة الثانية تنفيذها باقي التعليمات المطلوبة من نقطة التوقف السابقة ولتحقيق ذلك كان لابدّ من تمييز التعليمات قبل وبعد نقطة التوقف ضمن حالات مستقلة عن بعضها رغم تشاركها بالغاية التنفيذية للعمل. أما إذا افترضنا وجود حالة تنفيذية واحدة لكافة التعليمات المطلوبة فإنه سيتم دوماً الانتقال إليها وهذا سيؤدي إلى عدة مشاكل في مخطط الحالة لأنه قد توجد بعض التعليمات التي تنقل المهمة إلى حالة أخرى قبل نهاية تعليمات التنفيذ ولدى متابعة المهمة العمل بعد هذا الانتقال المفاجئ وعودتها إلى حالة التنفيذ الوحيدة الموجودة ضمن العملية فإنها ستُكرر التعليمات المنفذة سابقاً وهذا غير صحيح والأدق هو متابعتها فقط

التعليمات التي لم يسبق تنفيذها، لذا كان لابدّ من اعتبار عدة وضعيات للتنفيذ مستقلة وفق ما تقتضيه الحاجة.

يتضمن ملف الخرج الناتج على برنامج نظامي مكتوب بلغة الـ IF قابل للتنفيذ المباشر دون أي تعديل وهو يُكافئ الخرج المتوقع. نتيجة لتطبيق الأدوات التي تتيحها لغة IF (الشكل 4-12) على ملف الخرج السابق حصلنا على ملف جديد (الشكل 4-13) بامتداد (aut.*) يُمثّل المحاكاة الشاملة (Exhaustive Simulation) لكافة الحالات الممكنة والانتقالات الموافقة للإشارات المتبادلة ضمن النظام المقابل للتطبيق المدروس.



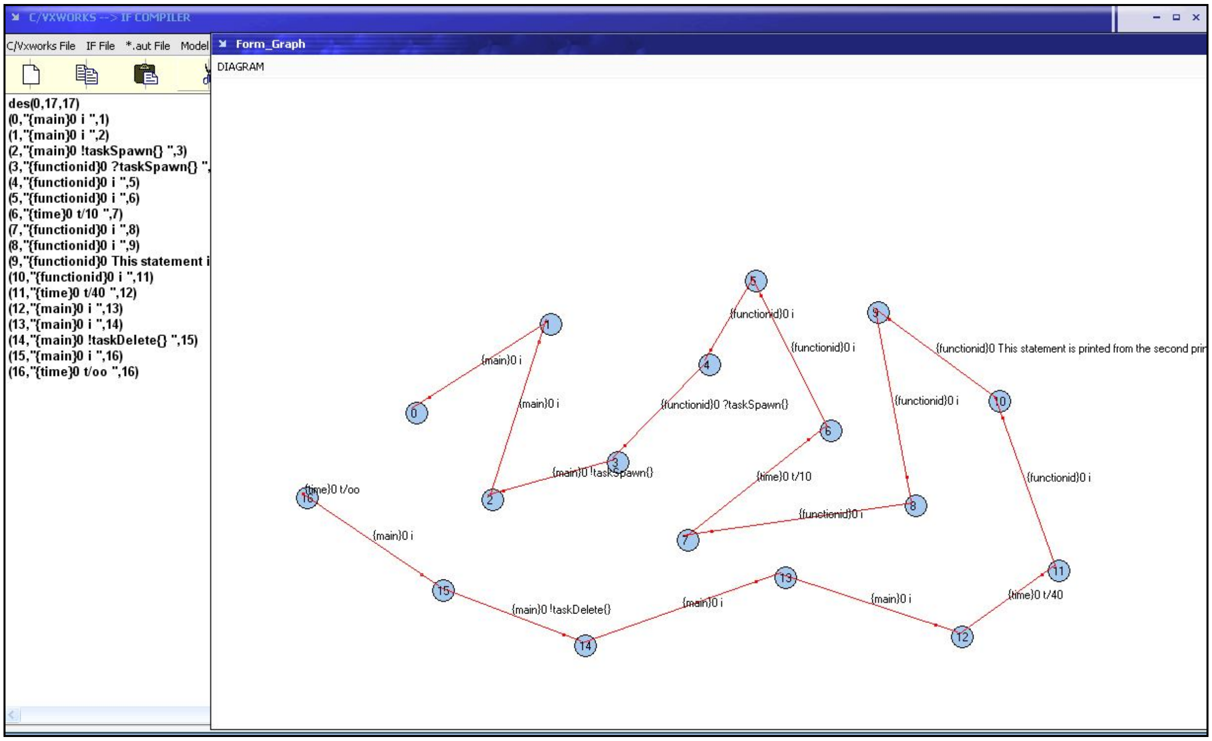

**الشكل (4-12): تطبيق أدوات IF على ملف الخرج**

يتضمن المخطط الناتج على ١٧ مناقلة و ١٧ حالة ويحتوي المناقلات التالية:

```
des(0,17,17)
(0,"{main}0 i ",1)
(1,"{main}0 i ",2)
(2,"{main}0 !taskSpawn{} ",3)
(3,"{functionid}0 ?taskSpawn{} ",4)
(4,"{functionid}0 i ",5)
(5,"{functionid}0 i ",6)
(6,"{time}0 t/10 ",7)
(7,"{functionid}0 i ",8)
(8,"{functionid}0 i ",9)
(9,"{functionid}0 This statement is printed from the second print_statement function  ",10)
(10,"{functionid}0 i ",11)
(11,"{time}0 t/40 ",12)
(12,"{main}0 i ",13)
(13,"{main}0 i ",14)
(14,"{main}0 !taskDelete{} ",15)
(15,"{main}0 i ",16)
(16,"{time}0 t/oo ",16)
```

**الشكل (4-13): ملف الحالات الشامل المكافئ لبرنامج IF**

إن حصولنا على ملف الحالات المكافئ للبرنامج المكتوب وفق IF (الذي هو خرج مترجمنا) يؤكد صحة تعليمات التحويل المعتمدة ودقة النتائج المعطاة وفق المترجم. يتم عادة معالجة هذا الملف وفق عدد من المراحل (كما سنشرح لاحقاً) للحصول على المخطط (Graph) البياني (الشكل 4-14) المقابل لملف الحالات النهائي.




**الشكل(4-14): مخطط الأتومات المقابل لملف المحاكاة الشامل**

## 5.4 الخاتمة:

لقد قمنا خلال هذا الفصل بإيضاح التعليمات المتقابلة بين لغة C/VxWorks ولغة IF وكيف تم توصيفها وفق صيغ الأتومات وبرمجتها بلغة IF، كما قمنا باختبار المترجم النهائي للتأكد من تكامليته وجودة ووثوقية عمله وتحديد مدى دقة وصحة نتائجه مما يسمح باعتباره أول مترجم يتناول تعليمات أنظمة الزمن الحقيقي بهذا الأسلوب يُتيح مقابلتها بتعليمات لغة أخرى خاصة تُمكننا من التحقق من صحة القيود الزمنية المفروضة على هذه الأنظمة.

يكون عادة المخطط المقابل لملف المحاكاة الشامل ضخماً جداً ويحتوي عدداً هائلاً من المناقلات والحالات بحيث يُقابل كل قيد زمني في مخطط الحالات الشامل مسار ما يصعب تحديده مباشرة، وبما أن الهدف الرئيسي من بحثنا هذا هو إتاحة عملية التحقق المباشر من صحة القيود الزمنية المفروضة على نظام الزمن الحقيقي المدروس لذا فيجب معالجة هذا الملف بحيث نحافظ فقط على الحالات والمناقلات المؤثرة في القيد الزمني المُراقب وهذا ما سنسعى لتحقيقه خلال الفصل القادم.

# ٥

# الفصل الخامس

# التحقق من القيود الزمنية

1.5 مقدمة.

2.5 المعالجة البرمجية للأتومات.

3.5 الخاتمة.

# التحقق من القيود الزمنية

## 1.5 مقدمة:

إن المراحل المختلفة التي تم وفقها بناء مترجمنا خلال الفصول السابقة جعلت عملية ترجمة التعليمات المطلوبة ممكنة ومتاحة بشكل كامل لكن ذلك لا يحقق مباشرة ما نسعى إليه، كما أن هدفنا من هذا البحث لا يكتمل بإيجاد المترجم فقطوا إنما وجوده هو خطوة وسيطة علينا توظيفها في التحقق الفعال والمباشر من صحة القيود الزمنية وبالشكل الذي يوفر علينا الكثير من الجهد البرمجي المبذول في حال غياب مثل هذا المترجم.

شر حنا سابقاً كيفية الحصول على مخطط الحالات (ملف aut.*) المقابل لبرنامج لغة IF (خرج مترجمنا) ووضحنا ما يحتويه من كم هائل من الإشارات الحتمية الإجبارية لعمل النظام لكنها غير مؤثرة في تقييم قيوده الزمنية، لذا كانت عملية التحقق من صحة القيود الزمنية تتطلب مراقبة إشارات وعمليات معينة موصّفة لهذه القيود ضمن النظام، فالسؤال الذي يطرح نفسه والذي نسعى للإجابة عليه الآن هو "كيف يمكننا قراءة هذا الأتومات الضخم للتأكد من صحة قيد معين؟".

إن قيامنا بذلك يقتضي معالجة ملف الحالات الشامل بحيث نحافظ فقط على الإشاراتالفعّالة في القيد الزمني المطلوب التأكد من صحته ولقد كان من المفترض لنا خلال هذه الخطوة استخدام تقنية CADP (التي تم ذكرها في الفصل الثاني) لتقوم باختصار الأتومات وفق معايير فلترة خاصة وتقديمه كملف نصي متاح للقراءة المباشرة كما تتيح معالجته بأسلوب بياني يساعد في توضيح القيد الزمني المطلوب دراسته بشكل أفضل، غير أن عدم مقدرتنا على تأمين هذه البرمجية غير المفتوحة المصدر أضاف متطلبات جديدة للبحث وكان لابدّ من برمجة هذا الجزء الإضافي من المهام التي تؤكد لنا صحة نتائج بحثنا بشكل أدق. سنوضح الآن خطوات المعالجة الإضافية التي قمنا بها لتحقيق ذلك.

## 2.5 المعالجة البرمجية للأتومات:

بما أن أي قيد زمني في نظام زمن حقيقيٍ ُعبَّر عنه بمسار معين ضمن مخطط الأتومات المكافئ لملف المحاكاة الشامل المقابل لبرنامج IFالموصّف للنظام المدروس، فإننا للقيام بعملية التحقق من صحة هذا القيد الزمني وتحديد مدى التقيد به (احترامه أو تجاوزه)، علينا معالجة هذا الملف الشامل بالأسلوب الذي يسمح لنا بمراقبة المسار المقابل للقيد المطلوب مباشرةوبالشكل الذي يساعدنا على تحقيق الهدف النهائي

من هذا البحث فنحن فعلياً غير مهتمين بالشكل الحالي للأتومات وإنما علينا إيجاد طريقة عملية لإبراز المناقلات المطلوبة.

من أجل القيام بعملية المراقبة المطلوبة قمنا بكتابة خوارزمية خاصة تأخذ مناقلات ملف المحاكاة الشامل (كما هو موضح في الفصل الأول، الشكل 1−2) كدخل لها وتُطبق عليها معايير فلترة محددّة بملف خارجي يحتوي العناصر المطلوب مراقبتها والقابلة للتعديل من قبل المستخدم (الملف بامتداد obs.*). فعلياً يحتوي ملف المحاكاة الشامل على كافة المناقلات الممكنة للنظام التي يُوصَّف كالتالي:

- يُكتب السطر الأول من هذا الملف وفق الصيغة: des(number1,number2,number3)

حيث يُمثل number1 رقم الحالة الابتدائية للمخطط، number2 يُمثل عدد المناقلات ضمن المخطط (عدد أسطر الملف−١ وتُقابل عدد الأسهم ضمن المخطط البياني)، بينما يُمثل number3 عدد الحالات الكلية (عدد عقد المخطط).

- أما باقي الأسطر (عددها يساوي عدد المناقلات) فهي تُوصّف دوماً وفق التالي:

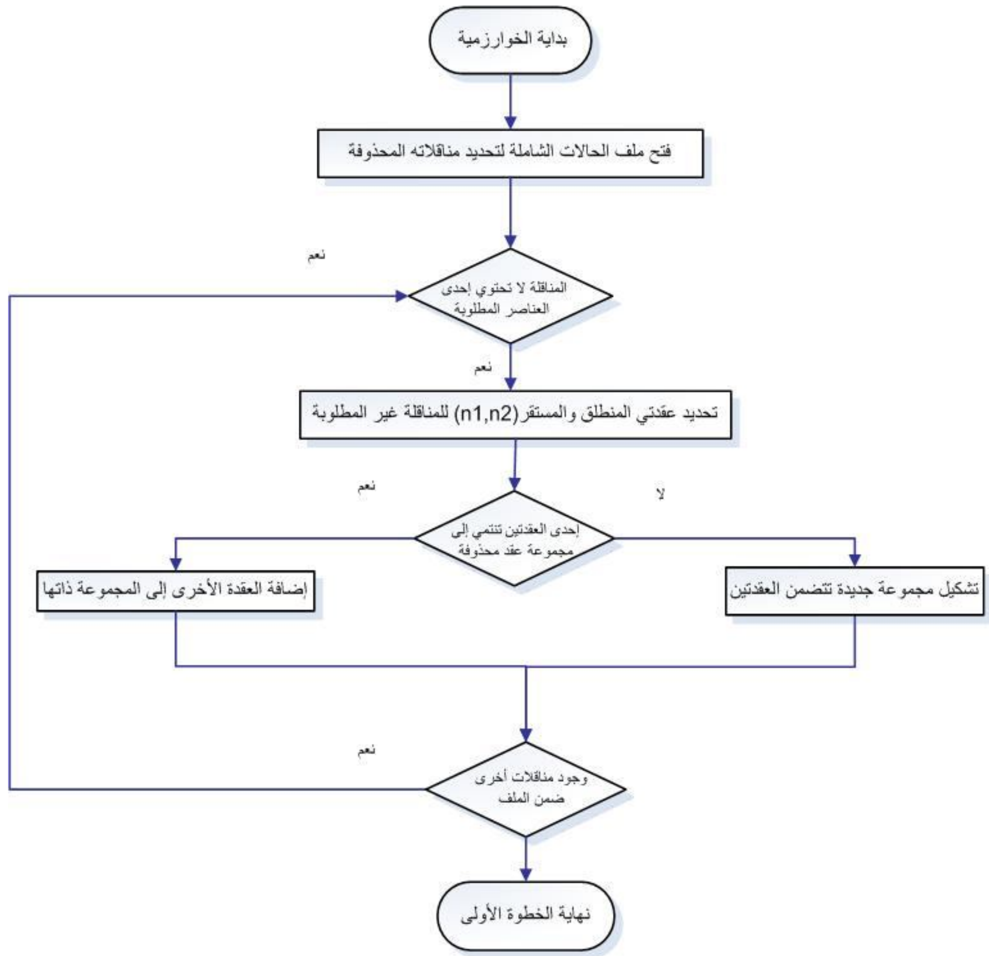
(From_node,"description_transition",To_node)

تُمثل From_node عقدة منطلق المناقلة، تُمثل To_node العقدة المستقر، أما توصيف المناقلة فتُحدد بـ description_transition وتتضمن كافة الإشارات والعناصر المتفاعلة معاً التي أدت لحدوثها.

### 1.2.5 خوارزمية المراقبة:

تظهر الصعوبة الأساسية لعملية المراقبة في حذف المناقلات غير المطلوبة (التي لا تؤثر في القيد المُراقب). فعلياً المناقلات المحذوفة هي التي لا تحتوي في توصيفها على أحد العناصر المحددة ضمن ملف المراقبة (obs.*) المكتوب من قبل المستخدم والمتضمن للإشارات أو العمليات المتحكمة بصحة القيد الزمني المدروس. إن حذف المناقلة يتم معه حذف عقدتي الطرفين من المخطط العام (عقدة المنطلق والمستقر)، وبالتالي إذا احتوت إحدى المناقلات المطلوبة (التي تدخل في المسار المُراقب) على عقدة محذوفة فإن علينا استبدالها بعقدة بديلة مناسبة وبالشكل الذي يحافظ على الارتباط السليم بين المناقلات. المشكلة التي صادفتنا هنا هي احتمال وجود عدة عقد انتقالية متتالية محذوفة معاً ففي مثل هذه الحالة يصعب علينا تحديد العقدة المكافئة مباشرة وإنما يتم استبدال العقدة المحذوفة بالمجموعة المكافئة لها المُحددة وفق خوارزمية المراقبة التالية التي تتضمن خطوتي معالجة.

- **الخطوة الأولى:** يتم وفقها تصفح ملف المحاكاة الشامل لتحديد المناقلات غير المطلوبة (الواجب حذفها) وتشكيل مجموعات العقد المحذوفة المقابلة لها (الشكل 5-1).




**الشكل(5-1): مخطط الخطوة الأولى لخوارزمية المراقبة**

نلاحظ أن كل مناقلة غير مطلوبة تتم معالجتها وفق الخطوات التالية:

- تحديد عقدتي المنطلق والمستقر للمناقلة قيد المعالجة.
- اختبار انتماء العقدتين إلى مجموعات العقد المحذوفة:

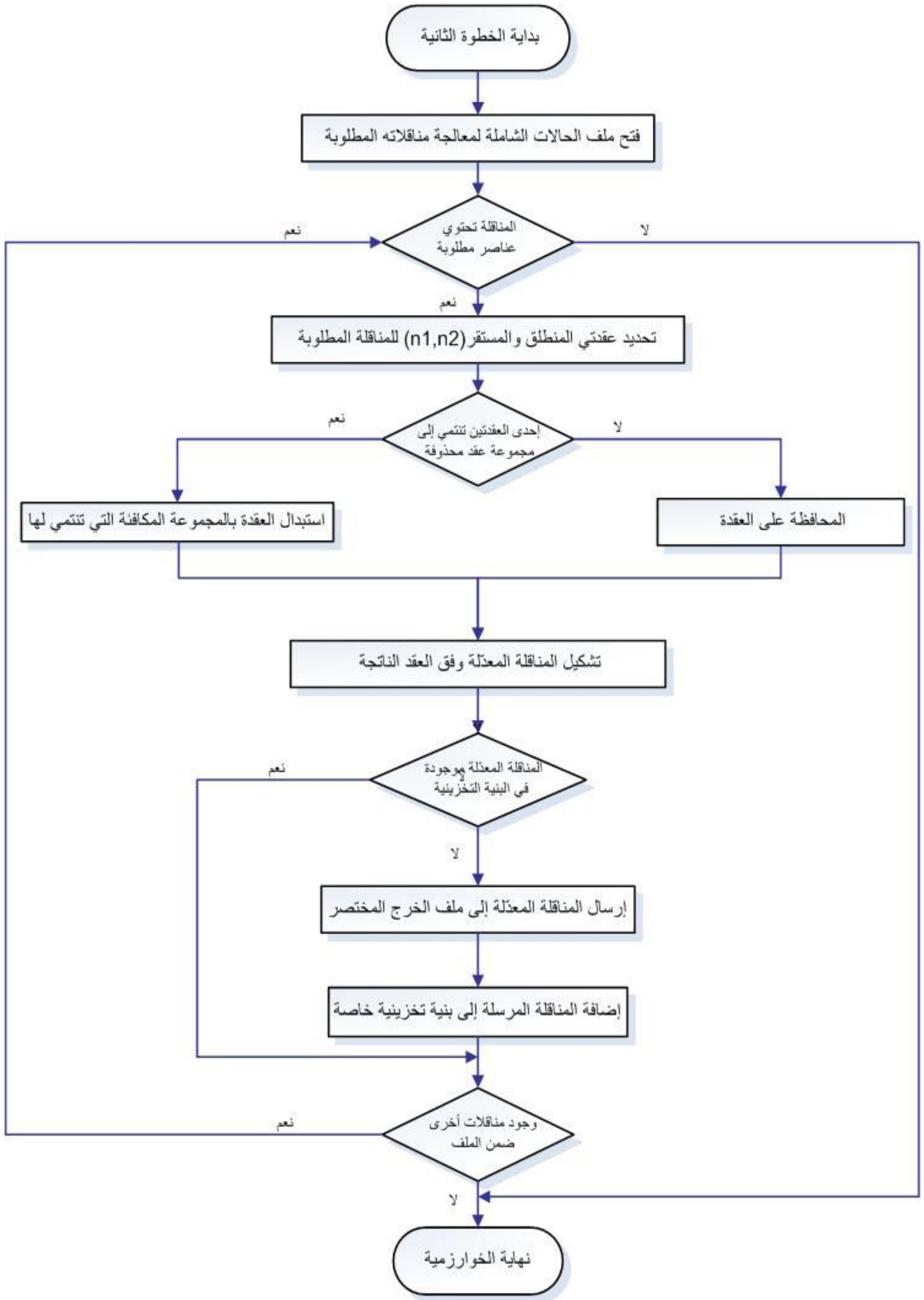
✓ حيث يتم تشكيل مجموعة جديدة ذات اسمٍ مميّزٍ (حُدّد برمجياً بشكل فريد وغير متكرر) في حال عدم انتماء كلتا العقدتين إلى أي مجموعة، هذه العملية تتم أيضاً عند بداية المعالجة لعدم وجود أي مجموعات مشكّلة بعد.

✓ في حال انتماء إحدى العقدتين إلى مجموعة مشكّلة مسبقاً يتم إضافة العقدة الأخرى إلى المجموعة ذاتها.

- لا يتم اتخاذ أي إجراء لدى وجود كلتا العقدتين ضمن إحدى المجموعات المحذوفة.

• **الخطوة الثانية:** تتضمن هذه المرحلة (الشكل 5-2) عملية إرسال المناقلات غير المحذوفة إلى الملف الناتج عنها بعد معالجة عقدتي الطرفين وتعديلهما بالشكل المناسب، حيث يتم تعديل عقدة طرفية لمناقلة

مطلوبة إذا كانت تنتمي إلى إحدى مجموعات العقد المحذوفة عند ذلك تُستبدل هذه العقدة باسم المجموعة المكافئة التي تنتمي إليها ثم تُرسل المناقلة المعدّلة إلى ملف الخرج النهائي الناتج عن هذه المعالجة.




**الشكل(5-2): مخطط الخطوة الثانية لخوارزمية المراقبة**

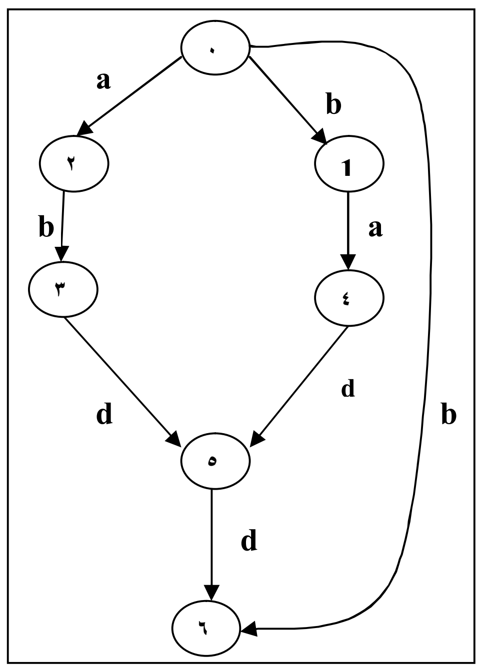
يتم تكرار هاتين الخطوتين على كافة مناقلات ملف الدخل حتى تصل إلى نهايته لينتج معنا ملف الخرج المطلوب الذي سيتم معالجته بيانياً لرسمه لاحقاً . المشكلة التي صادفتنا لدى التطبيق العملي لهذه

الخوارزمية هي عدم اختزالها للمناقلات المعدّلة الناتجة ذات الواصفات المتماثلة التي تكون فيها القيم (عقدة المنطلق، توصيف المناقلة، عقدة المستقر) مشتركة بين أكثر من مناقلة ففي هذه الحالة تُعتبر المناقلة مكررة ويجب إرسالها إلى ملف الخرج مرة واحدة فقط. لأخذ ذلك بعين الاعتبار تم تنفيذ خطوة مقارنة إضافية تُطبّق على المناقلة المعدّلة قبل إرسالها، حيث يتم حفظ كافة المناقلات المرسلة إلى ملف الخرج ضمن بنية تخزينية خاصة وفي كل مرة قبل القيام بعملية إرسال المناقلة المعدّلة الجديدة إلى الملف تُختبر هذه المناقلة فيما إذا كانت موجودة مسبقاً ضمن البنية التخزينية أم لا، لأن وجودها يدل على إرسالها للملف خلال معالجة سابقة وبالتالي يُمنع تكرار عملية إرسالها ثانية لملف الخرج وإنما يتم متابعة معالجة المناقلة التالية لها ضمن ملف الدخل. لقد تم برمجة هذه الخوارزمية المعدّلة بالتعليمات المناسبة، ثم قمنا باختبارها وفق عدد من التطبيقات العملية البسيطة للتحقق من صحة نتائجها وأنها ستعطي مناقلات معدّلة صحيحة مهما كانت محتويات ملف الدخل والحالات المتضمنة فيه.

- **تطبيق عملي لخوارزمية المراقبة:** بفرض لدينا الأتومات التالي (الشكل 5-3) المؤلف من (6) حالات و(8) مناقلات والمقابل لملف الدخل (الشكل 5-4) المكافئ لبرنامج مكتوب وفق لغة IF (لا تهمنا التفاصيل البرمجية للمثال وإنما ذُكر هنا لتوضيح خوارزمية المراقبة فقط).

```
des(0,8,6)
(0,"b",1)
(0,"a",2)
(0,"b",6)
(1,"a",4)
(2,"b",3)
(4,"d",5)
(3,"d",5)
(5,"d",6)
```

**الشكل(5-4): ملف الدخل المطلوب مراقبته**




**الشكل(5-3): مخطط الأتومات**

لدى تطبيق خوارزمية المراقبة السابقة على ملف الدخل وبعد تحديد الإشارات (a,d) ضمن ملف المراقبة، نلاحظ وجود عدد من المناقلات التي سيتم حذفها من الملف النهائي (لعدم احتواء توصيفها على إشارة مطلوبة للمراقبة) وتُعالج وفق الترتيب التالي:

- المناقلة (0,"b",1) تؤدي إلى تشكيل أول مجموعة محذوفة A={0,1}.
- المناقلة (0,"b",6) وفيها إحدى عقدتي الطرفين تنتمي إلى المجموعة السابقة A لذا تُضاف العقدة الأخرى إلى المجموعة A لتصبح هذه المجموعة تتضمن العناصر A={0,1,6}.
- المناقلة (2,"b",3) تؤدي إلى تشكيل مجموعة عقد محذوفة جديدة لعدم انتماء عقدتيها إلى أي مجموعة مشكلة سابقاً وبالتالي تنتج لدينا المجموعة B={2,3}.

أي أن العقد المحذوفة الناتجة عن تطبيق الخطوة الأولى صُنِّفت ضمن مجموعتين:

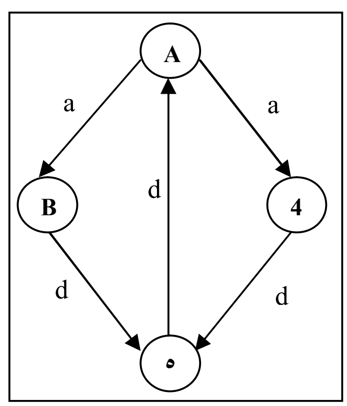
A={0,1,6} , B={2,3}

أما تطبيق الخطوة الثانية للخوارزمية فيؤدي إلى تحديد المناقلات المقابلة للإشارات المطلوبة، التي تنتج من معالجة ملف الدخل وتعديل مناقلاته وفق الترتيب التالي:

- المناقلة (0,"a",2) تُستبدل ضمنها العقدة (2) بمجموعتها المحذوفة B المقابلة لها والعقدة (0) تُستبدل بالمجموعة A لتصبح المناقلة بعد التعديل وفق الصيغة: (A,"a",B) .
- المناقلة (1,"a",4) تُستبدل ضمنها العقدة (1) بالمجموعة المكافئة لها A لتنتج المناقلة (A,"a",4).
- المناقلة (4,"d",5) لا تتم أي عملية تعديل عليها لعدم وجود أي من عقدتيها ضمن المجموعات المحذوفة وبالتالي تبقى المناقلة على حالها وتُرسل مباشرة إلى ملف الخرج.
- المناقلة (3,"d",5) تُستبدل ضمنها العقدة (3) بالمجموعة المكافئة لها B لتنتج المناقلة (B,"d",5).
- المناقلة (5,"d",6) تُستبدل ضمنها العقدة (6) بالمجموعة المكافئة لها A لتنتج المناقلة المعدّلة: (5,"d",A). بالتالي ينتج لدينا ملف الخرج التالي (الشكل 5-5) المؤلف من (4) حالات و(5) مناقلات الذي يقابله الأتومات الموضح في (الشكل 5-6).

des(0,5,4)
(A ,"a",B)
(A,"a",4)
(4,"d",5)
(B,"d",5)
(5,"d",A)

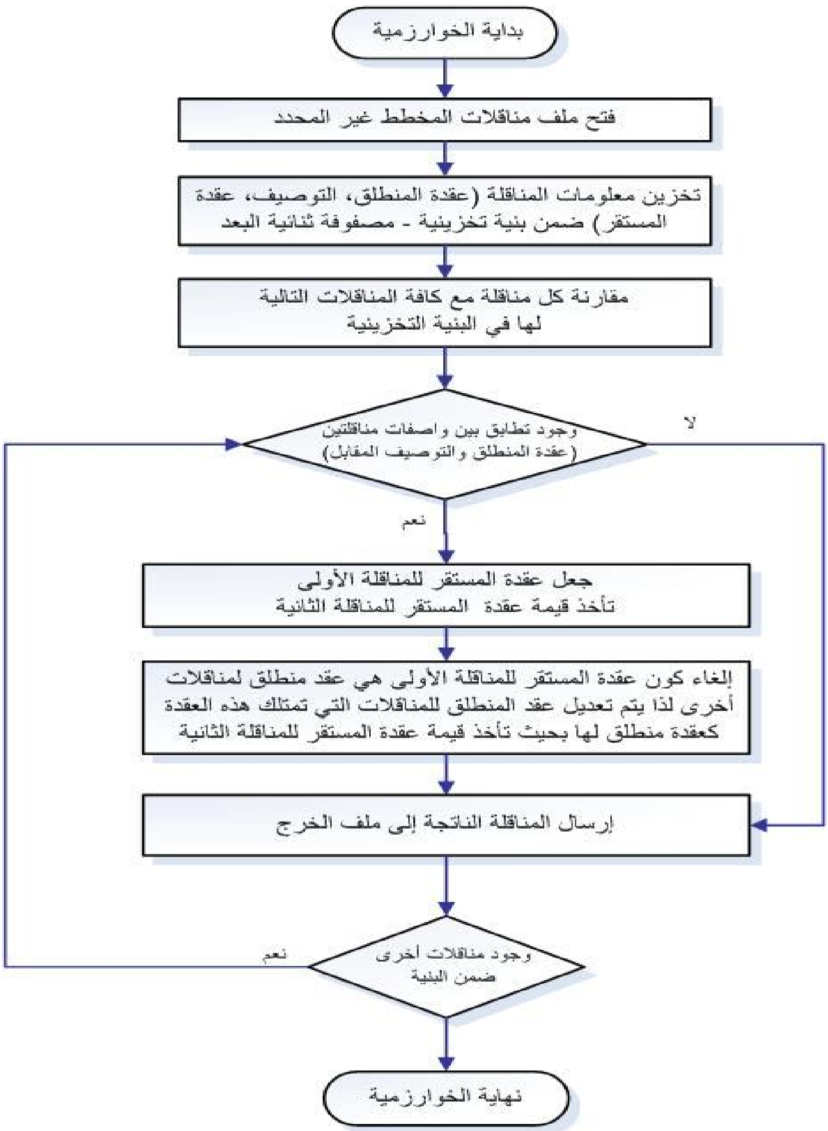
**الشكل(5-5): الملف الناتج عن المراقبة**




**الشكل(5-6): مخطط الأتومات المراقب**

### 2.2.5 تحسين مناقلات الأتوماتاتالمُ راقب:

إن الملف الذي ينتج عن خوارزمية المراقبة السابقة غالباً ما يتضمن مناقلات تؤدي إلى غموض في الأتومات المقابل له (AmbiguityDiagram) وعدم تحديدهُعر ّف المخطط غير المحدد بأنه المخطط الذي يحتوي مناقلات غير حتمية (Undeterministic) يمكن وفقها للحدث نفسهانطلاقاً من حالة واحدة أن يؤدي للانتقال إلى حالتين مستقلتين إضافة إلى إحتمال وجود بعض المناقلات المتطابقة. قمنا بمعالجة ذلك وفق خوارزمية إزالة الغموض (الشكل 5-7) التي يتم من خلالها دمج عقدتي المستقر للمناقلتين المؤديتين إلى الغموض واستبدالهما بعقدة واحدة لها تسمية إحداهما بحيثتُ عتبر هذه العقدة الناتجة عن الدمج عقدة منطلق لكافة المناقلات الصادرة عن هاتين العقدتينمعاً ، كما يتم حذف تكرار المناقلات المتطابقةوأخيراً إعادة ترقيم حالات المناقلات النهائية للملف الناتج.




**الشكل(5-7): مخطط خوارزمية إزالة الغموض**

بتطبيق خوارزمية إزالة الغموض على الملف الموضح في الشكل (5-5) والناتج عن عملية مراقبة العناصر (a,d) ضمن التطبيق العملي المذكور في الفقرة السابقة. نلاحظ وجود المناقلتين (A,"a",B)، (A,"a",4) اللتين تؤديان إلى عدم تحديد الأتومات المقابل ولدى معالجتهما يتم دمج عقدتي المستقر بالعقدة (4) واستبدال كافة عقد المنطلق ذات القيمة (B) للمناقلات بالعقدة (4)، حيث يصبح ملف الأتومات المحدد كالتالي:

des(0,4,٣)
(A,"a",4)
(4,"d",5)
(٤,"d",5)
(5,"d",A)

لتنتهِ خطوات تحسين المخطط بعد، لابدّ من حذف تكرار المناقلات المتطابقة ليصبح الملف النهائي الناتج عن إزالة الغموض هو التالي:

des(0,٣,٣)
(A,"a",4)
(4,"d",5)
(5,"d",A)

أخيراً قد نقوم بإعادة ترقيم حالات المناقلات الناتجة من خلال استعراض كافة المناقلات الموجودة في الملف الأخير واستبدال كل عقدة طرفية برقم تسلسلي تزداد قيمته لدى كل مصادفة جديدة لرقم عقدة غير معالج مسبقاً مع ضرورة التقيد باستبدال العقدة بالرقم ذاته في كافة المناقلات. تبدأ عملية إعادة الترقيم هذه بمقابلة العقدة الطرفية للمناقلة الأولى بالرقم (0) (العقدة التي نراها أولاً تُعطى الرقم الأصغر ي) وهكذا نتابع عملية الترقيم بحيث تُستثنى العقدة المعالجة من المقارنات اللاحقة. بتطبيق عملية الترقيم هذه على الملف الأخير نحصل على الملف النهائي (الشكل 5-8) الناتج عن عملية التحسين وإعادة الترقيم الذي يتضمن المناقلات التالية:

des(0,٣,٣)
(0,"a",1)
(١,"d",2)
(2,"d",٠)

**الشكل(5-8): الملف الناتج عن تحسين الأتومات**

تُعد عملية إعادة الترقيم هذه خطوة تكميلية غير ضرورية برمجياً وإنما تتم للمحافظة على ملف المحاكاة وفق صيغته العامة ليسهل معالجته والقيام برسمه بيانياً. تُتاح هذه العملية ضمن تطبيقنا كخطوة اختيارية للمستخدم ويمكن بالعودة إلى الملحق (III) للحصول على معلومات تفصيلية عن الواجهة التي تُمكّننا من القيام بهذه العملية.

- **تقييم صحة الخوارزمية:** هناك بعض الخوارزميات والنظريات الجاهزة التي تُعالج موضوع الأتومات المنتهي غير المحدد NFA(Nondeterministic Finite Automata) وكيفية تحويله إلى أتومات

محدد DFA(Deterministic Finite Automata) مكافئ يقبل نفس اللغة ويحتوي على انتقال وحيد لكل حالة من حالاته عند كل رمز من رموز الأبجدية. إحدى الخوارزميات الأساسية في هذا المجال خوارزمية تشكيل المجموعات الفرعية (subset construction algorithm) [34]، التي تتعامل مع الأتومات على أنه خماسية (Q,∑,∂,q0,F) حيث:

- Q هي مجموعة الحالات $Q=\{q_0,q_1,\dots,q_n\}$ وعددها محدود .
- ∑ هي أبجدية الدخل (مجموعة منتهية).
- $q_0$ هي الحالة الأولية أو الابتدائية للأتومات.
- F هي مجموعة الحالات النهائية وهي مجموعة جزئية من Q.
- δ هو تابع الانتقال الجديد المعرّف كالتالي: $Q*\Sigma\rightarrow 2^Q\,\delta$:

$$\delta(q,a)=\{p_1,\dots p_k\} \in Q$$

تعتمد هذه الخوارزمية على تشكيل جميع الحالات الممكنة انطلاقاً من الحالات الأصلية وبجميع الأطوال (فردية، ثنائية، ثلاثية،...) وتستثني الحالات المعزولة التي لا يمكن الوصول إليها انطلاقاً من الحالة البدائية لعدم وجود انتقالات عندها. أي يتم وفق هذه الخوارزمية الانطلاق من الحالة البدائية مع إيجاد الانتقالات المقابلة لها وكلما نتجت حالة جديدة نعتمدها ونعاود إيجاد انتقالاتها بحيثُ حسب تابع الانتقال الموافق للحالة المركّبة عند كل حالة من حالاتها الضمنية لتنتج حالة مركبة جديدة وهكذا تتكرر المعالجة إلى أن تشمل كافة الحالات المتاحة. لقد قمنا باستخدام هذه الخوارزمية للتحقق من صحة النتائج التي تعطيها خوارزميتنا المقترحة وتم مقارنة الأتومات الناتج عنها مع الأتومات المحدد الناتج عن خوارزميتنا فكانا متطابقين وفق نفس النتائج. على سبيل المثال بتطبيق هذه الخوارزمية على المثال العملي الموضح سابقاً واستخدامها لإزالة غموض الأتومات الموضح في (الشكل 5-5)، نجد أن التابع يُعطى بالجدول التالي:

| δ | a | d |
|---|---|---|
| $q_0$ | $q_1$ | ∅ |
| $q_1$ | ∅ | $q_2$ |
| $q_2$ | ∅ | $q_0$ |

⟺

| δ | a | d |
|---|---|---|
| A | [B,4] | ∅ |
| [B,4] | ∅ | 5 |
| 5 | ∅ | A |

**الجدول(5-1): التابع الناتج عن الخوارزمية**

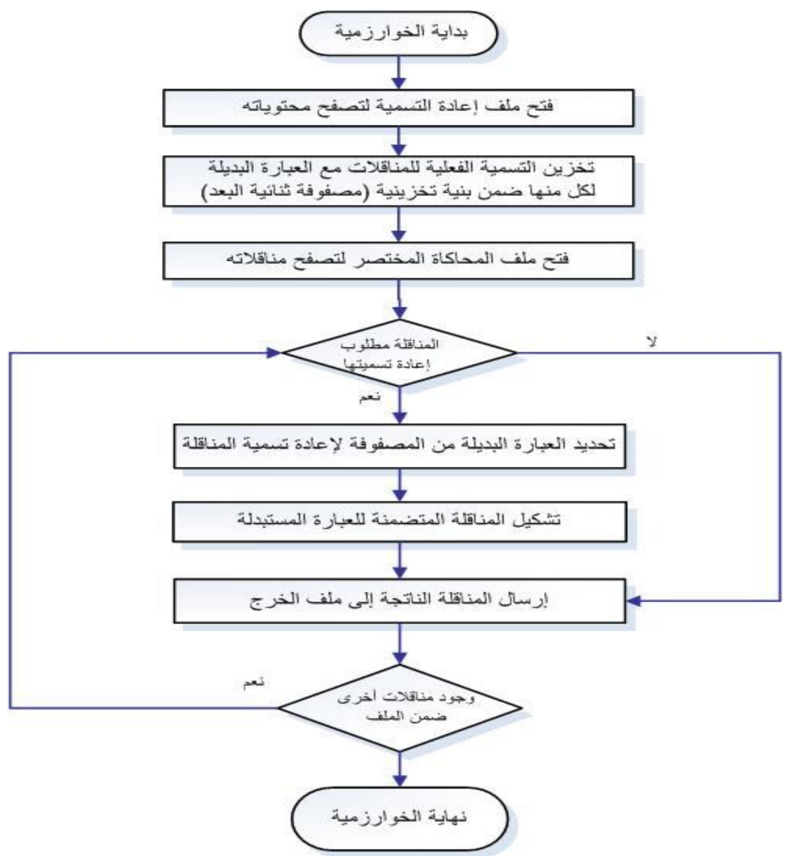
يمكننا نصياً التعبير عن الأتومات الناتج عن خوارزمية تشكيل المجموعات الفرعية كالتالي:

($q_0$,"a", $q_1$)
($q_1$,"d", $q_2$)
($q_2$,"d", $q_0$)

وهذا يتطابق تماماً مع مناقلات الأتومات الناتج عن خوارزمية إزالة الغموض المقترحة سابقاً وبالتالي فإن نتائجها صحيحة.

### 3.2.5 خوارزمية إعادة التسمية:

أخيراً من أجل التوضيح البياني الأدق للمسار المُراقب قمنا بإتاحة إمكانية رسم الأتومات المقابل للملف الناتج عن المعالجات السابقة بعد إعادة تسمية مناقلاته، أي أننا أضفنا ميزة جديدة اختيارية يستطيع المستخدم من خلالها تحديد عبارات بديلة لتوصيف المناقلات بمصطلحات خاصة أكثر دقةً وتعبيراً للمهمة المرتبطة بها. غالباً ما يتضمن توصيف المناقلة على عبارة شاملة لكافة الإشارات والعمليات المشاركة في الانتقال المرفق بهذه المناقلة، التي يمكن أن تؤدي أثناء الرسم إلى تشويش المخطط وعدم وضوحه. إن تطبيق عملية إعادة التسمية الموضحة في (الشكل 5-9) على المناقلات قبل الرسم البياني للأتومات المقابل لها يجعل هذه المناقلات أكثر تعبيراً ووضوحاً أثناء الرسم.




**الشكل(5-9): مخطط خوارزمية إعادة تسمية المناقلات**

نقوم أولاً بكتابة ملف خارجي يتضمّن عبارات محددة للمناقلات المطلوب استبدالها مكتوبة بصيغة خاصة تحترم قواعد إعادة التسمية المصطلحة والتي لابدّ من الإلتزام بها أثناء كتابته، ليتم لاحقاً فتح هذا الملف

وتصفحه وتحديد مناقلاته مع بدائلها وتخزينها وفق بنية مصفوفة تُعتمد في عملية إعادة التسمية. يتم ضمن هذا الملف كتابة التوصيف الفعلي الخاص بكل مناقلة (اسم المناقلة) مع العبارة البديلة له والفصل بين هاتين العبارتين بالرمز (%) كما نستخدم الرمز السابق ذاته للدلالة على نهاية السطر. أما عملية تحديد التسمية البديلة للمناقلة فيمكن أن تتم بأي أسلوب مع الأخذ بعين الاعتبار أننا لدى ذكر الإشارات المتبادلة وفق المناقلة نقوم بذكر اسم العملية الفعالة في إشارة المناقلة المدروسة يتبعها أقواس تتضمن الإشارات المستقبلة من قبل العملية المحددة منتهية بالرمز (R_) والإشارات المرسلة تنتهي بالرمز (S_) وهذا مجرد اصطلاح محدد من قبلنا فقط لصياغة ملف إعادة التسمية وتسهيل معالجته برمجياً. بفرض قمنا بمتابعة معالجة مناقلات الملف النهائي الموضح في (الشكل 5-8)، بحيث تم تحديد المعلومات التي يتضمنها ملف إعادة التسمية بالشكل التالي:

```
a% this transition equivalents a %
d% this transition equivalents d %
```

إن الملف النهائي الناتج عن عملية إعادة التسمية يتضمن المناقلات التالية:

```
des(0,٣,٣)
(0,"this transition equivalent a",1)
(١,"this transition equivalent d",2)
(2,"this transition equivalent d",٠)
```

علماً أن كافة خطوات المعالجة السابقة التي أتحناها لتحسين عملية الرسم البياني تم دعمها من خلال واجهات رسومية مرئية سهلة الاستخدام والتطبيق من قبل المستخدم ويبقى فقط عليه الإلتزام الصحيح بتسلسل مراحل المعالجة أثناء التطبيق العملي. كما أننا ضبطنا ذلك برمجياً من خلال عدد من الواجهات الحوارية التوضيحية التي تم شرحها ضمن الملحق (II) والتي تُنبّه المستخدم على أي تجاوزات أو أخطاء قد تحدث أثناء الاستخدام نتيجة السرعة في العمل أو جهله بالاستخدام الصحيح للتطبيق.

## 3.5 الخاتمة:

استعرضنا في هذا الفصل كيفية معالجة مخطط الحالات المقابل للنظام الناتج عن مترجمنا و المكتوب وفق لغة IF وكيف يتم مراقبته وفق معايير خاصة وتحسينه بإلغاء المناقلات المتطابقة فيها وإزالة الغموض منه، مما يمكننا من الحصول على الملف النهائي المتضمن فقط للعناصر المقابلة للقيد الزمني المراقب. شرحنا أيضاً كيف أتحنا إمكانية رسم مخطط الأتومات المقابل للملف النهائي مع القدرة على إعادة تسمية مناقلاته وفق رغبة المستخدم بالشكل الذي يجعلها أكثر تعبيراً وتوضيحاً للقيد الزمني المدروس ويجعل من السهل علينا تتبعه مباشرة وتحديد مدى احترامه أو اختراقه.

# ٦

# الفصل السادس

# تطبيقات عملية شاملة

1.6 مقدمة.

2.6 تطبيقات توضيحية.

3.6 تطبيقات شاملة للتحقق من صحة القيود الزمنية.

4.6 مناقشة النتائج.

5.6 الخاتمة.

# تطبيقات عملية شاملة

## 1.6 مقدمة:

قدّمنا خلال الفصول السابقة لمحة شاملة عن مراحل بحثنا وكيف أنه أصبح بإمكاننا التحقق من صحة القيود الزمنية الموصّفة ضمن برامج C/VxWorks من خلال تطبيق خطوات المعالجة المختلفة سابقة الذكر. سنحاول في هذا الفصل التحقق من صحة عملنا بشكل كامل وفق عدد من التطبيقات البسيطة والشاملة لكافة مراحل البحث.

## 2.6 تطبيقات توضيحية:

سنوضح الآن بعض التطبيقات العملية التي قمنا باختبارها للتأكد من صحة تعليمات المقابلة المعتمدة بين اللغتين وتحقيقها للهدف البرمجي المنشود.

### 1.2.6 عمل السيمافور الثنائي:

التطبيق التالي هو برنامج مكتوب بلغة C/VxWorks (ملف الدخل الشكل 6-1) يحتوي على ثلاث مهام: تقوم المهمة الأساسية Main بخلق وتفعيل المهمتين first_task و second_task اللتين تتبادلان عملية تحرير وحجز السيمافور sem_available ذاته أثناء قيامهما بالأعمال المطلوبة منهما.

```
#include <vxworks.h>
#include<tasklib.h>
#include<time.h>
#include<stdio.h>
#include <math.h>
#include <semlib.h>
#include <string.h>
#include <stdlib.h>
LOCAL int firsttaskid;
LOCAL int secondtaskid;
LOCAL SEM_ID sem_available;
void firstFunc(void);
void secondFunc(void);
int main (void)
{	sem_available = semBCreate(SEM_Q_FIFO,SEM_EMPTY);
	firsttaskid=taskSpawn("first_task",20,0,200,(FUNCPTR)firstFunc,0,0,0,0,0,0,0,0,0,0);
	secondtaskid=taskSpawn("second_task",20,0,200,(FUNCPTR)secondFunc,0,0,0,0,0,0,0,0,0,0);
	printf(" Program tests the semaphore working \n");
	return 0;
}
void firstFunc(void)
{	semTake(sem_available,WAIT_FOREVER);
```

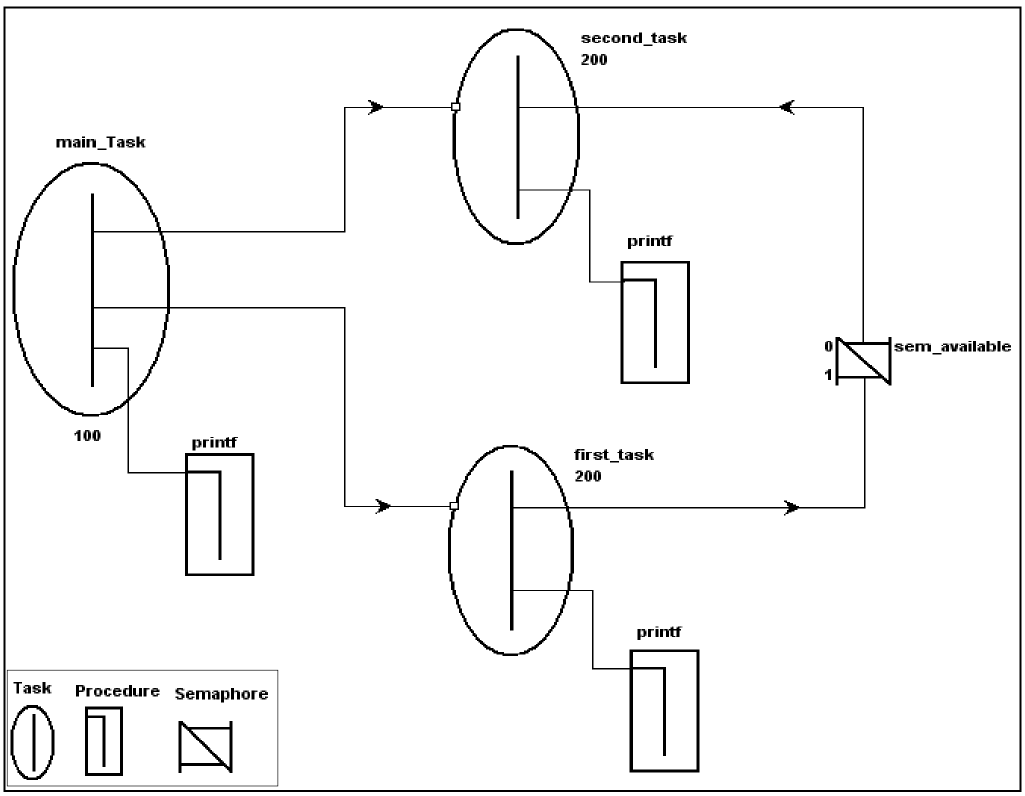
```
        printf(" The first task restrained semaphore by semTakeRoutine \n");    }
void secondFunc(void)
{
        semGive(sem_available);
        printf("The second task released semaphore by semGiveRoutine  \n");
}
```

**الشكل (6-1): البرنامج وفق لغة C/VxWorks**

إن تنفيذ هذا البرنامج وفق نظام VxWorks يعطي العبارتين التاليتين:

- ✓ The second task released semaphore by semGiveRoutine
- ✓ The first task restrained semaphore by semTakeRoutine

وهذا يتوافق مع كون المهمة الأولى لا تستطيع متابعة عملها إلا بعد أن يتم تحرير السيمافور المطلوب من قبل المهمة الثانية (الحاجزة له). مخطط برنامج الدخل المقابل لهذا البرنامج موضح وفق (الشكل 6-2).




**الشكل (6-2): مخطط برنامج الدخل باستخدام Lacatre**

إن برنامج الخرج المكتوب وفق لغة IF والناتج عن ترجمة البرنامج السابق، مُثّل نظاماً يتألف من عدد من الإشارات المتبادلة بين العمليات (firsttaskid,secondtaskid,sem_available,main) وبمعالجة هذا البرنامج نحصل على ملف المحاكاة الشامل الذي يحتوي (157) حالة و(368) مناقلة. وبما أننا نسعى خلال هذا المثال إلى إثبات صحة مناقشتنا لأفكار السيمافور ودقة تحديدنا للتعليمات المقابلة له وفق لغة IF فإن ذلك يقتضي مراقبة عناصر معينة (إشارات مرسلة أو مستقبلة، عمليات، عبارات خاصة) ضمن ملف المحاكاة الشامل المكافئ لبرنامج خرج المترجم. تم تحديد العناصر المطلوبة للمراقبة ضمن ملف خارجي

يُعالجبرمجياً ليتم من خلاله مراقبة ملف المحاكاة والمحافظة فقط على مناقلات العناصر المحددة في ملف المراقبةالذي تضمّن مايلي:

- semGive!: عملية إرسال الإشارة semGive من قبل secondtaskid لتحرير السيمافور.
- semGive?: استقبال الإشارة semGive من قبل السيمافور sem_available ليتم معالجتها وفق وضعه الحالي.
- semBTake!: إرسال الإشارة semBTake من قبل العملية firsttaskid لطلب الحجز.
- semBTake?: استقبال الإشارة semBTake من قبل السيمافور sem_available.
- Ack_SemTake!: إرسال السيمافور إشارة توكيد للعملية الطالبة للحجز تدل على إتاحته لها.
- Ack_SemTake?: استقبال العملية الطالبة للحجز إشارة التوكيد من السيمافور لتنجز مهامها.
- restrained: إحدى مفردات العبارة الدالة على حجز السيمافور، تطبعها العملية firsttaskid.
- released: إحدى مفردات العبارة الدالة على تحرير السيمافور التي يتم طباعتها من قبل العملية secondtaskid.

نتيجة لمراقبتنا هذه العناصر نحصل على الملف التالي (الشكل 6-3) الذي يحتوي على (15) حالة و(16) مناقلة فقط.

```
des(0,16,15)
(0,"{secondtaskid}0 !semGive{{secondtaskid}0} {secondtaskid}0 The second task released semaphore by semGiveRoutine ",1)
(0,"{firsttaskid}0 !semBTake{{firsttaskid}0,-1} ",2)
(1,"{sem_available}0 ?semGive{{secondtaskid}0} ",3)
(2,"{sem_available}0 ?semBTake{{firsttaskid}0,-1} ",4)
(1,"{firsttaskid}0 !semBTake{{firsttaskid}0,-1} ",5)
(2,"{secondtaskid}0 !semGive{{secondtaskid}0} {secondtaskid}0 The second task released semaphore by semGiveRoutine ",6)
(5,"{sem_available}0 ?semGive{{secondtaskid}0} ",7)
(3,"{firsttaskid}0 !semBTake{{firsttaskid}0,-1} ",7)
(6,"{sem_available}0 ?semBTake{{firsttaskid}0,-1} ",8)
(4,"{secondtaskid}0 !semGive{{secondtaskid}0} {secondtaskid}0 The second task released semaphore by semGiveRoutine ",8)
(7,"{sem_available}0 ?semBTake{{firsttaskid}0,-1} {sem_available}0 !Ack_SemTake{{sem_available}0} ",9)
(8,"{sem_available}0 ?semGive{{secondtaskid}0} {sem_available}0 !Ack_SemTake{{sem_available}0} ",10)
(9,"{firsttaskid}0 ?Ack_SemTake{{sem_available}0} ",11)
(10,"{firsttaskid}0 ?Ack_SemTake{{sem_available}0} ",12)
(11,"{firsttaskid}0 The first task restrained semaphore by semTakeRoutine ",13)
(12,"{firsttaskid}0 The first task restrained semaphore by semTakeRoutine ",14)
```

**الشكل (6-3): مخطط الحالات المراقب**

يوضح هذا الملف كافة الاحتمالات الممكنة لإرسال واستقبال الإشارات المطلوب مراقبتها حيثاعتباراً من الحالة الابتدائية(0) تكون المهمتان في وضعية جاهزية ويمكن لكلتيهما القيام بالإرسال وبالتالي نكون أمام أحد احتمالين إما أن تقوم المهمة secondtaskid بإرسال semGive أولاً أو أن تقوم firsttaskid بإرسال

إشارتها semBTakeأولاً وهكذا يتتالى تراسل الإشارات بين العمليات إلى أن نصل إلى إحدى حالات المخطط النهائيـة (13,14) أي أن هنـاك مسـارين مختلفين لعمـل هذا النظـام وكلاهمـا يؤكـد أن عمليـة تحريـر وحجـز السيمافور تتم بشكل صحيح يتوافق مع السيناريو المدروس.

❖ بتعديل المثال السابقوا ٍضافة تعليمتي (taskSuspend(firsttaskid), taskSuspend(firsttaskid للتأكد من أن عمليـةإرسـال و استقبال الإشـارات المقابلـة لهمـا تتم بشكل صـحيح، تصـبح التوابـع البرمجيـة المتأثرة بهذا التعديل وفق الشكل التالي (6-4):

```
int main (void)
{           sem_available = semBCreate(SEM_Q_FIFO,SEM_EMPTY);
            firsttaskid=taskSpawn("first_task",20,0,200,(FUNCPTR)firstFunc,0,0,0,0,0,0,0,0,0,0);
            taskSuspend(firsttaskid);
            secondtaskid=taskSpawn("second_task",20,0,200,(FUNCPTR)secondFunc,0,0,0,0,0,0,0,0,0,0);
            printf(" Program tests the semaphore working \n");
            return 0;
}
void secondFunc(void)
{           semGive(sem_available);
            taskResume(firsttaskid);
            printf("The second task released semaphore by semGiveRoutine  \n");
}
```

**الشكل (6-4): الشيفرة البرمجيةالمعدّلة**

إن ملـف المحاكـاة الشـامل المقابـل للنظـام النـاتج عن التعديل السـابق يتضـمن (319) حالـة و(833) مناقلة وبإجراء مراقبة للعناصر التالية:

(!taskSuspend, ?taskSuspend, !taskResume, ?taskResume, !Ack_SemTake, ?Ack_SemTake, restrained, released)

- taskSuspend!: تُعبّر عن إرسال إشارة لإيقاف عمل المهمة المنفذة.
- taskSuspend?: استقبال إشارة الإيقاف المؤقت من قبل المهمة المقصودة.
- taskResume!: تُمثّل إرسال إشارة المتابعة للمهمة.
- taskResume?: استقبال الإشارة لمتابعة عمل المهمة التي تم إيقافها سابقاً. ينتج لدينا عن عملية المراقبة ملف الحالات التالي (الشكل 6-5) الذي يتضمن (21) حالة و(26) مناقلة.

```
des(0,26,21)
(0,"{firsttaskid}0 ?taskSuspend{} ",1)
(0,"{secondtaskid}0 !semGive{{secondtaskid}0} {secondtaskid}0 !taskResume{} {secondtaskid}0 The second task released semaphore by semGiveRoutine  ",2)
(0,"{firsttaskid}0 ?taskSuspend{} ",3)
(2,"{firsttaskid}0 ?taskSuspend{} ",4)
(1,"{secondtaskid}0 !semGive{{secondtaskid}0} {secondtaskid}0 !taskResume{} {secondtaskid}0 The second task released semaphore by semGiveRoutine  ",4)
(4,"{firsttaskid}0 ?taskResume{} ",5)
(0,"{secondtaskid}0 !semGive{{secondtaskid}0} {secondtaskid}0 !taskResume{} {secondtaskid}0 The second task released semaphore by semGiveRoutine  ",6)
(2,"{firsttaskid}0 ?taskSuspend{} ",7)
```

```
(6,"{firsttaskid}0 ?taskSuspend{} ",8)
(3,"{secondtaskid}0 !semGive{{secondtaskid}0} {secondtaskid}0 !taskResume{} {secondtaskid}0 The second task released semaphore by semGiveRoutine  ",8)
(7,"{firsttaskid}0 ?taskResume{} ",9)
(8,"{firsttaskid}0 ?taskResume{} ",10)
(2,"{sem_available}0 ?semBTake{{firsttaskid}0,-1} {sem_available}0 !Ack_SemTake{{sem_available}0} ",11)
(6,"{sem_available}0 ?semGive{{secondtaskid}0} {sem_available}0 !Ack_SemTake{{sem_available}0} ",12)
(7,"{sem_available}0 ?semBTake{{firsttaskid}0,-1} {sem_available}0 !Ack_SemTake{{sem_available}0} ",13)
(11,"{firsttaskid}0 ?taskSuspend{} ",13)
(8,"{sem_available}0 ?semGive{{secondtaskid}0} {sem_available}0 !Ack_SemTake{{sem_available}0} ",14)
(12,"{firsttaskid}0 ?taskSuspend{} ",14)
(9,"{sem_available}0 ?semBTake{{firsttaskid}0,-1} {sem_available}0 !Ack_SemTake{{sem_available}0} ",15)
(13,"{firsttaskid}0 ?taskResume{} ",15)
(10,"{sem_available}0 ?semGive{{secondtaskid}0} {sem_available}0 !Ack_SemTake{{sem_available}0} ",16)
(14,"{firsttaskid}0 ?taskResume{} ",16)
(15,"{firsttaskid}0 ?Ack_SemTake{{sem_available}0} ",17)
(16,"{firsttaskid}0 ?Ack_SemTake{{sem_available}0} ",18)
(17,"{firsttaskid}0 The first task restrained semaphore by semTakeRoutine  ",19)
(18,"{firsttaskid}0 The first task restrained semaphore by semTakeRoutine  ",20)
```

**الشكل (6-5): مخطط الحالات المقابللمثال المعدّل**

يؤكد الملف السابق الموضح في (الشكل 6-5) أن معالجتنا للإشارتين taskSuspend وtaskResume (تم إيضاح أسلوب المعالجة في الفصل الرابع ضمن الفقرة "3.3.4 مشاكل في البنية الهيكلية القواعدية للغة IF") تتم بشكل صحيح ومناسب يتوافق مع الهدف البرمجي المنشود منهما. نلاحظ أن هذا الملف يحتوي مناقلات تؤدي إلى غموض المخطط المقابل (المناقلات المقصودة مميزة ضمن الشيفرة السابقة). على سبيل المثال نذكر المناقلتين اللتين تؤديان إلى حالة عدم التحديد:

```
(2,"{firsttaskid}0 ?taskSuspend{} ",4)
(2,"{firsttaskid}0 ?taskSuspend{} ",7)
```

وبمعالجة ذلك وفق خوارزمية إزالة الغموض نحصل على ملف المحاكاة المحدد المتضمن (15) حالة و(19) مناقلة وهو يحتوي المناقلات الموضحة في (الشكل 6-6) التالي:

```
des(0,19,15)
(0,"{firsttaskid}0 ?taskSuspend{} ",1)
(0,"{secondtaskid}0 !semGive{{secondtaskid}0} {secondtaskid}0 !taskResume{} {secondtaskid}0 The second task released semaphore by semGiveRoutine  ",2)
(2,"{firsttaskid}0 ?taskSuspend{} ",3)
(1,"{secondtaskid}0 !semGive{{secondtaskid}0} {secondtaskid}0 !taskResume{} {secondtaskid}0 The second task released semaphore by semGiveRoutine  ",3)
(3,"{firsttaskid}0 ?taskResume{} ",4)
(2,"{sem_available}0 ?semBTake{{firsttaskid}0,-1} {sem_available}0 !Ack_SemTake{{sem_available}0} ",5)
(2,"{sem_available}0 ?semGive{{secondtaskid}0} {sem_available}0 !Ack_SemTake{{sem_available}0} ",6)
(3,"{sem_available}0 ?semBTake{{firsttaskid}0,-1} {sem_available}0 !Ack_SemTake{{sem_available}0} ",7)
(5,"{firsttaskid}0 ?taskSuspend{} ",7)
(3,"{sem_available}0 ?semGive{{secondtaskid}0} {sem_available}0 !Ack_SemTake{{sem_available}0} ",8)
(6,"{firsttaskid}0 ?taskSuspend{} ",8)
(4,"{sem_available}0 ?semBTake{{firsttaskid}0,-1} {sem_available}0 !Ack_SemTake{{sem_available}0} ",9)
(7,"{firsttaskid}0 ?taskResume{} ",9)
(4,"{sem_available}0 ?semGive{{secondtaskid}0} {sem_available}0 !Ack_SemTake{{sem_available}0} ",10)
(8,"{firsttaskid}0 ?taskResume{} ",10)
(9,"{firsttaskid}0 ?Ack_SemTake{{sem_available}0} ",11)
(10,"{firsttaskid}0 ?Ack_SemTake{{sem_available}0} ",12)
```

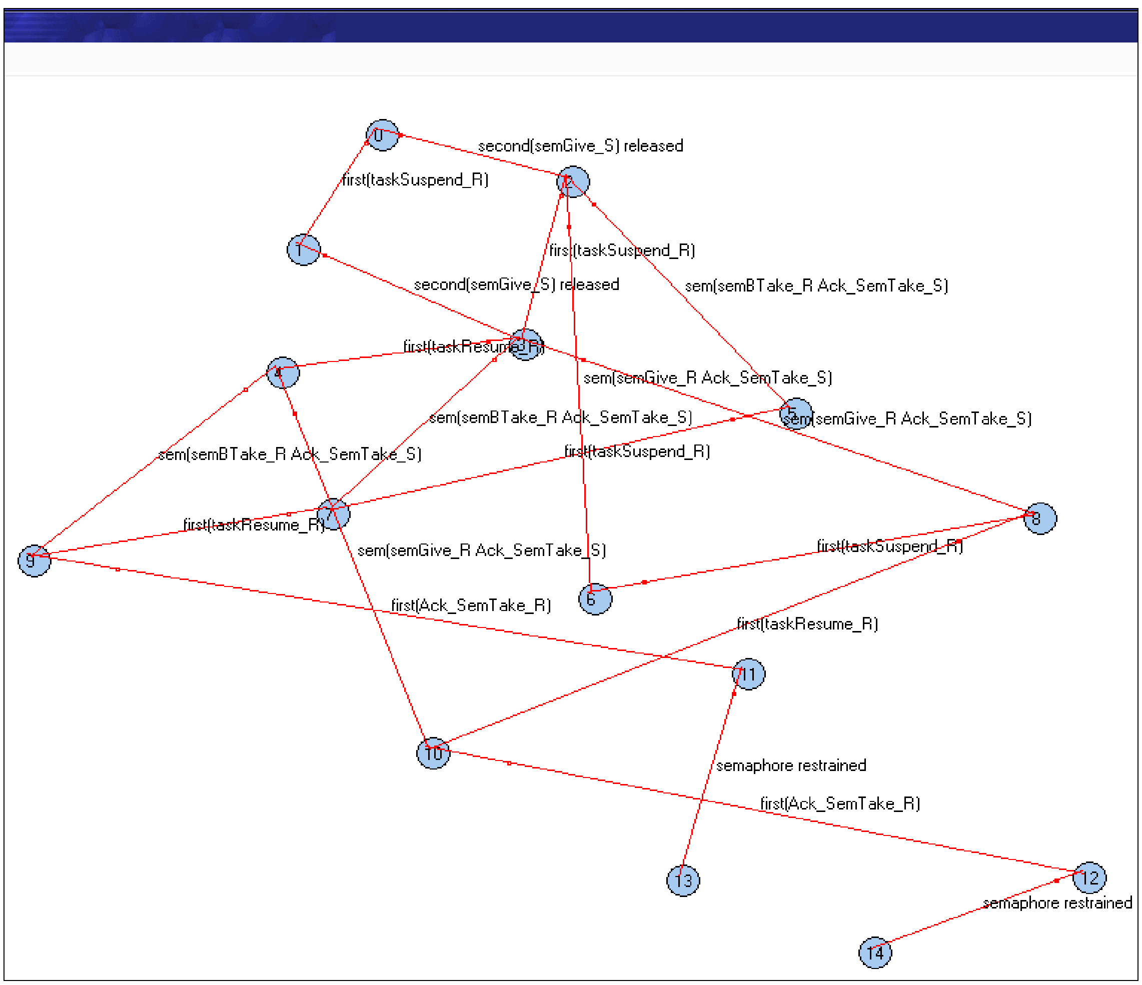
(11,"{firsttaskid}0 The first task restrained semaphore by semTakeRoutine ",13)
(12,"{firsttaskid}0 The first task restrained semaphore by semTakeRoutine ",14)

**الشكل (6-6): مخطط الحالات المحدد**

وبتطبيق خوارزمية إعادة التسمية على هذه المناقلات بحيث تم تحديد ملف إعادة التسمية وفق (الشكل 6-7) ينتج لدينا المخطط الموضح في (الشكل 6-8) الذي يتوافق مع الخرج المتوقع.

{firsttaskid}0 ?taskSuspend{} % first(taskSuspend_R) %
{secondtaskid}0 !semGive{{secondtaskid}0} {secondtaskid}0 !taskResume{} {secondtaskid}0 The second task released semaphore by semGiveRoutine % second(semGive_S) released %
{firsttaskid}0 ?taskResume{} % first(taskResume_R) %
{sem_available}0 ?semBTake{{firsttaskid}0,-1} {sem_available}0 !Ack_SemTake{{sem_available}0} % sem(semBTake_R Ack_SemTake_S) %
{sem_available}0 ?semGive{{secondtaskid}0} {sem_available}0 !Ack_SemTake{{sem_available}0} % sem(semGive_R Ack_SemTake_S) %
{firsttaskid}0 ?Ack_SemTake{{sem_available}0} % first(Ack_SemTake_R) %
{firsttaskid}0 The first task restrained semaphore by semTakeRoutine % semaphore restrained %

**الشكل (6-7): ملف إعادة التسمية**




**الشكل (6-8): مخطط الأتومات النهائي**

### 2.2.6 عمل المؤقت الزمني:

يتناول التطبيق التالي مثالاً بسيطاً عن المؤقت الزمني للتأكد من صحة معالجة تعليماته. هذا المثال عبارة عن جهاز لتحضير القهوة مرفق بمؤقت يُحدّد لحظة البدء بالتحضير ولحظة النهاية من أجل إصدار تنبيه للمستخدم بذلك وقد تم التعبير عنه وفق البرنامج التالي (الشكل 6-9):

```
#include <taskLib.h>
#include <time.h>
#include <stdio.h>
#include <math.h>
#include <semlib.h>
#include <vxWorks.h>
#include <wdLib.h>
LOCAL int coffeeId;
LOCAL WDOG_ID coffeeAlarmId;
void coffee(void);
void ALRCoffeeAlarm(void);
int main(void)
{         coffeeAlarmId=wdCreate();
          printf("coffee_alarm has been created \n");
          coffeeId=taskSpawn("CoffeeDrinking",255,0,200,(FUNCPTR)coffee,0,0,0,0,0,0,0,0,0,0);
          return(0);
}

void coffee(void)
{         wdStart(coffeeAlarmId,100,(FUNCPTR)ALRCoffeeAlarm,0);
          printf("Coffee alarm has been started \n");
}
void ALRCoffeeAlarm(void)
{         printf("Coffee alarm has finished \n");
}
```

**الشكل (6-9): البرنامج وفق C/Vxworks**

لدى معالجة برنامج لغة IF الناتج عن ترجمة البرنامج السابق نحصل على ملف المحاكاة الشامل الذي يتضمّن (44) حالة و(86) مناقلة. إن صحة معالجتنا تعليمات المؤقت الزمني تتحدد بانتقال هذا المؤقت وفق مراحله المختلفة من إنشاء وتفعيل إلى لحظة انتهائه العمل المُسند له، وللتأكد من ذلك قمنا بمراقبة كلمات من العبارات التالية التي تُقابل الإشارة wdCreate وwdStart المُعبّرة عن عمل المؤقت (الكلمات المراقبة محددة بنمط الخط الغامق):

(alarm has been **created**, Coffee alarm has been **started**, alarm has **finished**).

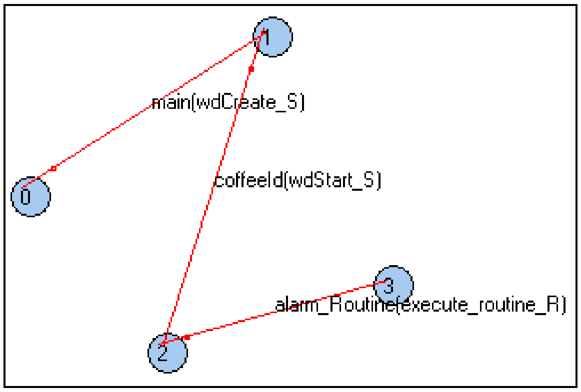
نتيجة لعملية المراقبة حصلنا على الملف التالي:

```
des(0,3,4)
(0,"{main}0 !wdCreate{} {main}0 !taskSpawn{} {main}0 coffee_alarm has been created  ",1)
(1,"{coffeeId}0 !wdStart{100} {coffeeId}0 Coffee alarm has been started  ",2)
(2,"{coffeeAlarmId_alarm_Routine}0 ?execute_routine{} {coffeeAlarmId_alarm_Routine}0 Coffee alarm has finished  ",3)
```

وبتطبيق خوارزمية إعادة التسمية على هذا الملف تصبح مناقلاته وفق الشكل التالي:

```
des(0,3,4)
(0," main(wdCreate_S) ",1)
(1,"coffeeId(wdStart_S) ",2)
(2,"alarm_Routine(execute_routine_R) ",3)
```

يؤكد المخطط البياني (الشكل 6-10) المقابل للمناقلات السابقة أن المؤقت الزمني يمر بمراحل عمله المحددة تماماً كما هو متوقع منه، أي أن كافة الإشارات المقابلة يتم إرسالها واستقبالها بشكل صحيح.




**الشكل (6-10): مخطط الملف النهائي**

❖ عند إجراء التعديل التالي (الشكل 6-11) على التابعين (coffee, ALRCoffeeAlarm) بحيث يترافق عمل المؤقت بسيمافور coffeeSemId خاص للتحكم بلحظة انتهاء عمله بدقة أكثر.

```
int main(void)
{       coffeeSemId = semBCreate(SEM_Q_FIFO,SEM_EMPTY);
        coffeeAlarmId=wdCreate();
        printf("coffee_alarm has been created \n");
        coffeeId=taskSpawn("CoffeeDrinking",255,0,200,(FUNCPTR)coffee,0,0,0,0,0,0,0,0,0,0);   return(0);
}
void coffee(void)  /*coffee Routine Body*/
{       wdStart(coffeeAlarmId,120,(FUNCPTR)ALRCoffeeAlarm,0);
        printf("Coffee alarm has been started \n");
        semTake(coffeeSemId,WAIT_FOREVER);
        printf("The coffee is drunk already \n");

}
void ALRCoffeeAlarm(void)
{       semGive(coffeeSemId);
        printf("Coffee alarm has finished \n");
}
```

**الشكل (6-11): توابعالبرنامج المعدّلة**

لدى تنفيذ البرنامج الناتج عن التعديل ضمن نظام VxWorks نتج لدينا الخرج التالي:

- ✓ Coffee alarm has been started
- ✓ Coffee alarm has finished
- ✓ The coffee is drunk already

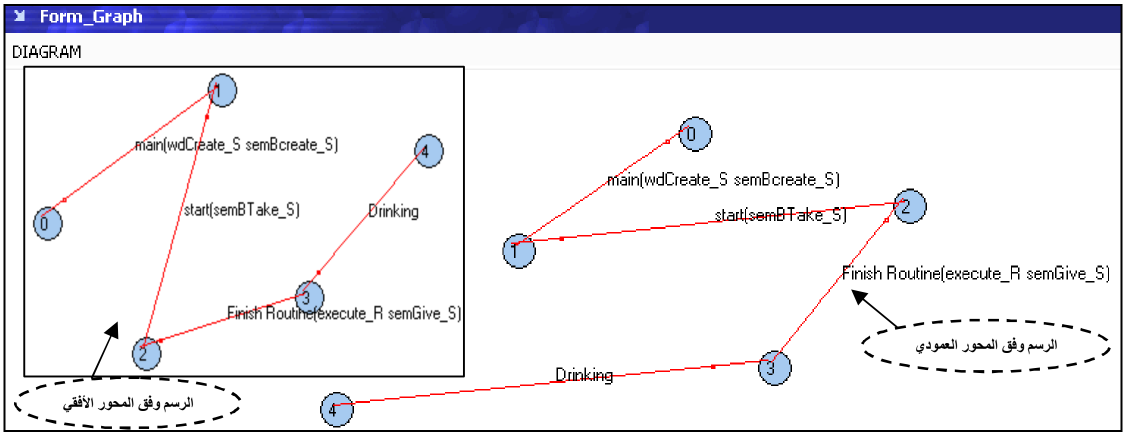
وبترجمة البرنامج المعدّل وفق مترجمنا ومعالجة خرجه بأدوات IF نحصل على ملف المحاكاة الشامل الذي يتضمن (77) حالة و(173) مناقلة. للتأكد من التوافق بين عمل السيمافور والمؤقت الزمني وصحة تعليماتهما قمنا بتحديد عناصر المراقبة (كلمات من العبارات) كالتالي:

(alarm has been **created**, alarm has been **started**, alarm has **finished**, The coffee is **drunk** already)

نتيجة لمراقبة هذه العناصر نحصل على الملف الذي يتضمن المناقلات التالية:

des(0,4,5)
(0,"{main}0 !semBcreate{{main}0,SEM_EMPTY} {main}0 !wdCreate{} {main}0 !taskSpawn{} {main}0 coffee_alarm has been created ",1)
(1,"{coffeeId}0 !wdStart{120} {coffeeId}0 Coffee alarm has been started {coffeeId}0 !semBTake{{coffeeId}0,-1} ",2)
(2,"{coffeeAlarmId_alarm_Routine}0 ?execute_routine{} {coffeeAlarmId_alarm_Routine}0 !semGive{{coffeeAlarmId_alarm_Routine}0} {coffeeAlarmId_alarm_Routine}0 Coffee alarm has finished ",3)
(3,"{coffeeId}0 The coffee is drunk already ",4)

بإعادة تسمية هذه المناقلات ورسم المخطط البياني المقابل نحصل على (الشكل 6-12). علماً أننا لتوضيح المخطط البياني بشكل أفضل أتحنا إمكانية الرسم وفق المحورين الأفقي والشاقولي وذلك يتحدد وفق رغبة المستخدم:




**الشكل (6-12): المخطط البياني النهائي**

حيث تضمن ملف إعادة التسمية المعلومات التالية:

{main}0 !semBcreate{{main}0,SEM_EMPTY} {main}0 !wdCreate{} {main}0 !taskSpawn{} {main}0 coffee_alarm has been created % main(wdCreate_S semBcreate_S) %
{coffeeId}0 !wdStart{120} {coffeeId}0 Coffee alarm has been started {coffeeId}0 !semBTake{{coffeeId}0,-1} % start(semBTake_S) %
{coffeeAlarmId_alarm_Routine}0 ?execute_routine{} {coffeeAlarmId_alarm_Routine}0 !semGive{{coffeeAlarmId_alarm_Routine}0} {coffeeAlarmId_alarm_Routine}0 Coffee alarm has finished % Finish Routine(execute_R semGive_S) %
{coffeeId}0 The coffee is drunk already % Drinking %

### 3.2.6 عمل علبة البريد:

لقد قمنا بالتحقق من صحة مقابلتنا لتعليمات علبة البريد وفق عدة تطبيقات عملية. على سبيل المثال نذكر التطبيق التالي (الشكل 6-13) الذي يتضمن مهمتين تقوم الأولى بإرسال رسالة معينة إلى صندوق البريد وتقوم المهمة الثانية باستلامها وا ٍظهارها وقد قمنا ببرمجة ذلك كالتالي:

```
#include <vxWorks.h>
#include <msgQLib.h>
#include <tasklib.h>
#define MESSAGE "Greetings from Task 1"
LOCAL int taskId;  LOCAL MSG_Q_ID  myMsgQId;
void taskMessage(void);
void main (void)
{
        myMsgQId = msgQCreate (10,100,MSG_Q_PRIORITY);
        printf(" message queue created \n");
        msgQSend (myMsgQId,MESSAGE,sizeof(MESSAGE),WAIT_FOREVER,MSG_PRI_NORMAL);
        printf("Normal priority message is sent it will block when queue is full  \n");
        taskId=taskSpawn("task2",20,0,200,taskMessage,0,0,0,0,0,0,0,0,0,0);
 }
void taskMessage(void)
{
        char msgBuf[100];
        msgQReceive(myMsgQId,msgBuf,100,WAIT_FOREVER);
        printf("get message from queue when necessary wait until msg is available \n");
        printf("The ReceivedMessage from task1 is %s\n", msgBuf);
        taskDelete(taskId);  }
```

**الشكل (6-13): برنامج علبة البريد**

لدى تنفيذ هذا البرنامج ضمن VxWorks نحصل على الخرج التالي:

- ✓ Normal priority message is sent it will block when queue is full
- ✓ get message from queue when necessary wait until msg is available
- ✓ The ReceivedMessage from task1 is Greetings from Task 1

باستخدام مترجمنا لترجمة تعليمات البرنامج السابق ومقابلتها بتعليمات لغة IF نحصل على البرنامج الذي يعطى لدى معالجته ملف المحاكاة المتضمن على(56) حالة و(94) مناقلة. غير أننا لاختبار صحة هذه التعليماتنا قمنا بتحديد العناصر التالية للمراقبة:

- msgQCreate!: إرسال إشارة تهيئة وبناء لعلبة البريد (مرسلة من قبل العملية main).
- msgQSend!: إرسال رسالة إلى علبة البريد (مرسلة من قبل العملية main).
- ack?: استقبال إشعار من علبة البريد myMsgQId للتأكيد أن الرسالة تم استقبالها بشكل صحيح.
- msgQReceive!: إرسال طلب استقبال رسالة من علبة البريد (مرسلة من قبل العملية taskId).
- msgQReceive?: استقبال علبة البريد myMsgQId للطلبية.
- msg_FirstWaiting?: استقبال العملية المنتظرة taskId للرسالة المطلوبة من علبة البريد.

نتيجة هذه المراقبة حصلنا على الملف التالي (الشكل 6-14):

```
des(0,11,9)
(0,"{main}0 !msgQCreate{10} {main}0 !taskSpawn{} {main}0 message queue created  {main}0 !msgQSend{-1} ",1)
(1,"{taskId}0 !msgQReceive{-1} ",2)
(1,"{main}0 ?ack{} {main}0 The Message: MESSAGE is sent successfully ",3)
(2,"{myMsgQId}0 ?msgQReceive{-1} {myMsgQId}0 !msg_FirstWaiting{} ",4)
(2,"{main}0 ?ack{} {main}0 The Message: MESSAGE is sent successfully ",5)
(3,"{taskId}0 !msgQReceive{-1} ",5)
(4,"{taskId}0 ?msg_FirstWaiting{} {taskId}0 The Message is received successfully ",6)
(4,"{main}0 ?ack{} {main}0 The Message: MESSAGE is sent successfully ",7)
(5,"{myMsgQId}0 ?msgQReceive{-1} {myMsgQId}0 !msg_FirstWaiting{} ",7)
(6,"{main}0 ?ack{} {main}0 The Message: MESSAGE is sent successfully ",8)
(7,"{taskId}0 ?msg_FirstWaiting{} {taskId}0 The Message is received successfully ",8)
```

**الشكل (6-14): ملف المحاكاة بعد تطبيق المراقبة**

وبتحديد ملف إعادة التسمية بحيث يتضمن المعلومات التالية:

```
{main}0 !msgQCreate{10} {main}0 !taskSpawn{} {main}0 message queue created  {main}0 !msgQSend{-1} % main(msgQCreate_S msgQSend_S)%
{taskId}0 !msgQReceive{-1}  % taskId(msgQReceive_S)%
{main}0 ?ack{} {main}0 The Message: MESSAGE is sent successfully % main(ack_R)sent successfully%
{myMsgQId}0 ?msgQReceive{-1} {myMsgQId}0 !msg_FirstWaiting{} % myMsgQId(msgQReceive_R msg_FirstWaiting_S) %
{taskId}0 ?msg_FirstWaiting{} {taskId}0 The Message is received successfully % taskId(msg_FirstWaiting_R)received successfully %
```

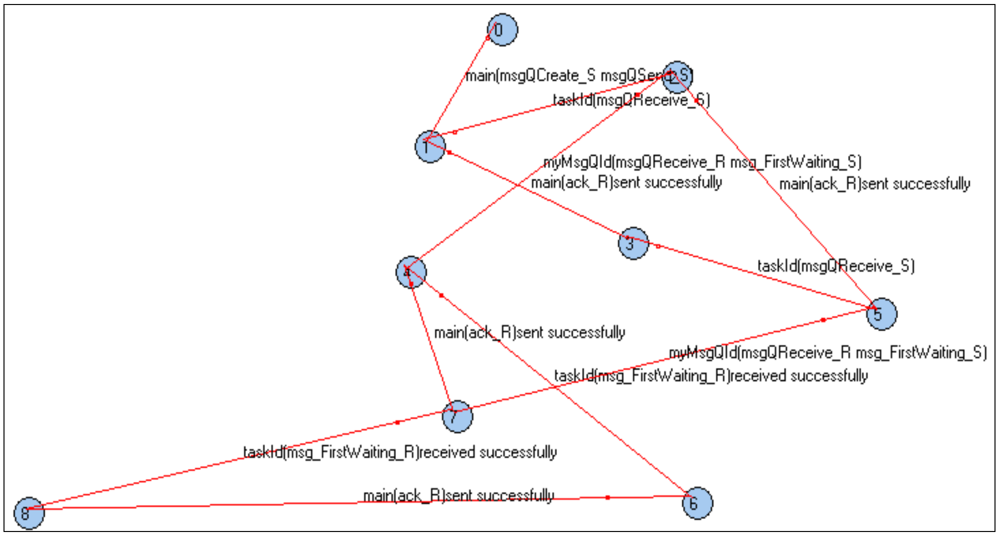
بتطبيق خوارزمية إعادة التسمية على الملف (الشكل 6-14) نحصل على الملف النهائي الذي يتضمن المناقلات (الشكل 6-15) التالية:

```
des(0,11,9)
(0," main(msgQCreate_S msgQSend_S)",1)
(1," taskId(msgQReceive_S)",2)
(1," main(ack_R)sent successfully",3)
(2," myMsgQId(msgQReceive_R msg_FirstWaiting_S) ",4)
(2," main(ack_R)sent successfully",5)
(3," taskId(msgQReceive_S)",5)
(4," taskId(msg_FirstWaiting_R)received successfully ",6)
(4," main(ack_R)sent successfully",7)
(5," myMsgQId(msgQReceive_R msg_FirstWaiting_S) ",7)
(6," main(ack_R)sent successfully",8)
(7," taskId(msg_FirstWaiting_R)received successfully ",8)
```

**الشكل (6-15): ملف المحاكاة النهائي**

يؤكد هذا الملف النهائي صحة معالجتنا لتعليمات علبة البريد. نلاحظ أن أول مناقلة ضمن هذا المخطط تتوافق مع إرسال العملية main للإشارة msgQCreate التي تؤدي إلى تهيئة علبة البريد وجعلها جاهزة لاستقبال الرسائل مع وضع رسالة ضمنها. ثم يكون أمامنا إحدى حالتين إما أن يتم مباشرة استقبال إشعار من علبة البريد بأن الرسالة استُلمت بشكل صحيح أو أن تقوم العملية الأخرى taskId بإرسال طلب لقراءة رسالة من العلبة (قبل استلام إشعار الاستلام الصحيح). تتناوب هاتان الحالتان ويتتابع إرسال الإشارات اللاحقة لهما وصولاً للحالة النهائية الأخيرة (8) التي تتوافق مع نهاية المسار الدال على الإرسال

الصحيح لعلبة البريد وقراءة الرسالة منها بشكل صحيح. المخطط البياني المقابل للملف النهائي (الشكل 6-15) تم توضيحه وفق (الشكل 6-16):



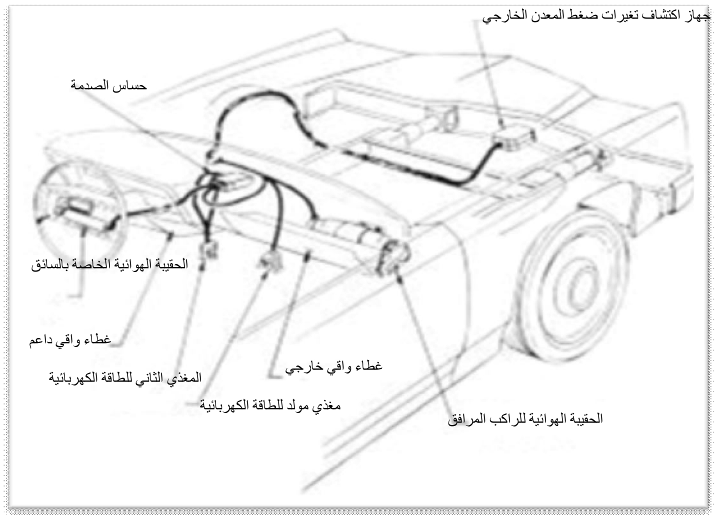

**الشكل (6-16): المخطط النهائي**

## 3.6 تطبيقات شاملة للتحقق من صحة القيود الزمنية:

### 1.3.6 تطبيق الحقيبة الهوائية:

سنُطبّق الآن اختبار عملي شامل لمراحل البحث المختلفة من أجل التأكّد من فعالية نتائجه وجودة استخدامه في التحقق من صحة القيود الزمنية ضمن أنظمة الزمن الحقيقي. لقد اعتمدنا في هذا على نظام زمن حقيقي يتناول عمل الحقيبة الهوائية (AirBag) المستخدمة لحماية السائق من الاصطدام بالزجاج الأمامي للسيارة عند تعرضه لأي حادث مفاجئ [35].

سنقوم الآن بتوضيح هذا النظام (الشكل 6-17) بشيء من التفصيل. يتألف نظام الحقيبة الهوائية من: الحقيبة، حساسات الصدمة، وحدة تحكم إلكترونية ECU (Electronic Control Unit) يُضاف إلى هذا النظام في بعض الأحيان مفتاح لإيقاف الحقيبة أو تفعيلها حسب رغبة السائق.




**الشكل(6-17): نظام الحقيبة الهوائية**

بشكل عام لن تؤدي كافة الصدمات بالضرورة إلى فتح الحقيبة (قد يحدث اصطدام بسيط غيرمؤذٍ بجدار ما نتيجة عدم انتباه السائق) إلا إذا ترافقت هذه الصدمة بسرعة معينة للسيارة (تتجاوز 30Km/h مثلاً)، عندها يُفعّل حساس الصدمة ليعطي إشارة للحقيبة الهوائية لكي تُفتح مؤديةً عملها في حماية السائق (الشكل 6-18) وفي حال تأخرها عن ذلك سيتم تحريك مقعد السائق نحو الخلف كخطوة حماية داعمة. علماً أن سرعة فتح الحقيبة تتعلق بسرعة تدفق الهواء ضمنها وتُقاس بـ ($m^3/sec$) وبحجم البالون المقاس بالـ ($m^3$)، يجب أن يكون زمن فتح الحقيبة أقل ... طدام رأس السائق بالزجاج بدءاً من لحظة الاصطدام.




**الشكل(6-18): نظام الحقيبة الهوائية**

إن تحديد قيم الزمن والسرعة التي تُنظم عملية فتح الحقيبة بشكل دقيق يقتضي منا محاكاة الصدمات التي تتعرض لها السيارة مع قياسات عملية لها، قد نلجأ إلى دُمية اختبار (الشكل 6-19) لتحديد الزمن الأعظمي المُتاح لفتح الحقيبة اعتماداً على سرعتها لحظة الاصطدام ومقدار الضرر الناجم عن ذلك مع مراعاة كافة السيناريوهات المحتملة للنظام.



الشكل(6-19): اختبار عملي لمحاكاة الصدمة

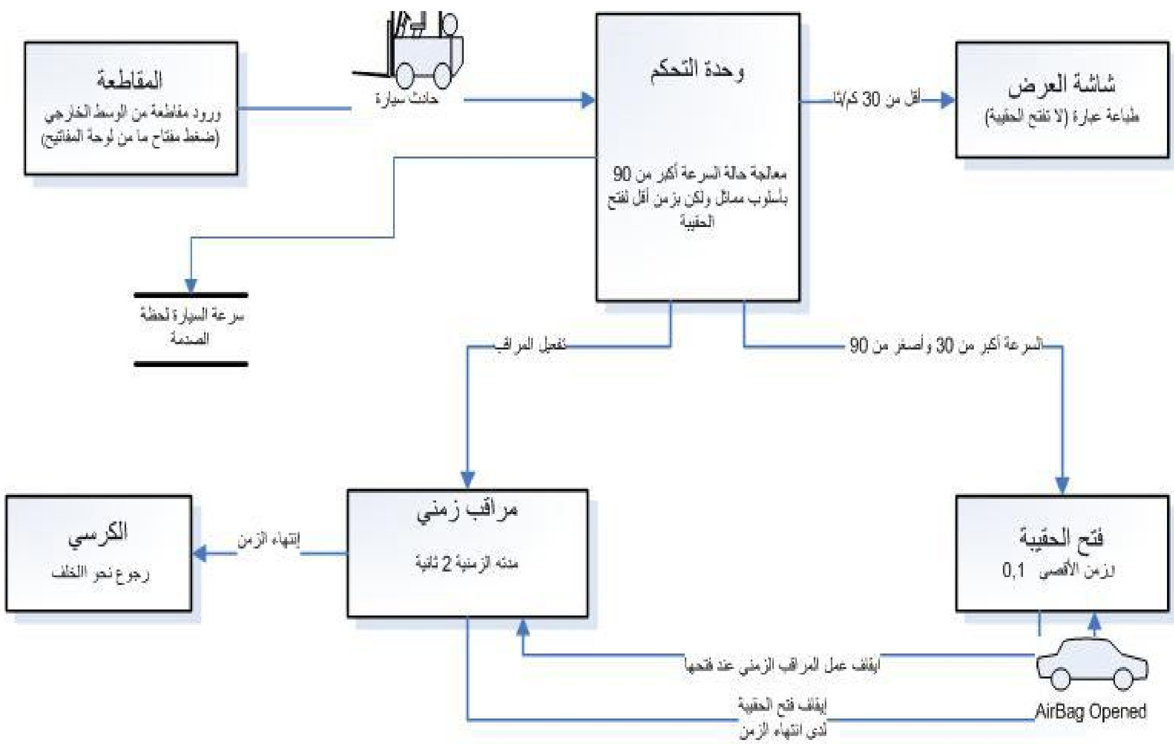
يُعتبر موضوع التحقق من صحة الزمن الحقيقي المستغرق لتنفيذ المهام المتضمنة وفق هذا التطبيق من أهم الأمور المعالجة التي تحتاج إلى دقة في التحديد إذ لا يمكننا معالجة مثل هذه المسألة بقيم افتراضية فحسبوا إنمالابدّ من الدراسة المنطقية والقياسات الصحيحة للزمن المستغرق لكل مهمة مع الأخذ بعين الاعتبار الأزمنة المحددة للمؤقتات المستخدمة ضمن هذا التطبيق، ولقد تم خلال هذا النظام توصيف مجموعة من القيود الزمنية لضبط العمل عند حدوث أي صدمة وافتراض معايير خاصةً نظّم أداء الحقيبة إذ لا يوجد توصيف ثابت لذلكوا إنما افترضنا الزمن الأقصى المتاح لفتح الحقيبة هو DeadLine=3sec وحجم بالون الحقيبة ($1m^3$) وأن هناك ثلاث حالات ممكنة لعمل هذا النظاماعتماداً على سرعة السيارة لحظة الاصطدام:

- سرعة السيارة أقل من (30Km/h): لنتُبدي الحقيبة الهوائية أي استجابة.
- سرعة السيارة بين (٣٠–٩٠): يجب أنتُفتح الحقيبة وفق سرعة معينة، بفرض سرعة انتفاخ البالون هي ($10m^3/sec$) عندها سيكون الزمن المستغرق لنفخه (1/10=0.1) والزمن المتبقي لفتح الحقيبة اعتباراً من لحظة الصدمة هو (3-0.1=2.9 sec)،علماً أن (الزمن=السرعة/الحجم)و أن كل ١٠٠٠ ميلي ثانية تقابل ثانية واحدة، وكل (18 ticks) تأخذ ثانية واحدة.
- سرعة السيارة تفوق (90Km/h): سيتطلب فتح الحقيبة بسرعة أكبر وسيتم تأمين ذلك بزيادة سرعة ضخ الهواء ضمن البالون فعلى فرض تم تحديد السرعة في هذه الحالة ($20m^3/sec$) عندها يكون الزمن اللازم لنفخ البالون فقط هو(0.05sec) والزمن المتبقي لفتح الحقيبة هو (3-0.05=2.95sec).

أما إذا لم تتم استجابة الحقيبة خلال الفترة الزمنية المحددة فإن ذلك يدل على اختراق القيد الزمني المفروض على عملها ويتطلب وجود ميكانيك رديف يضمن حماية السائق سيتممثلاً إرسال إشارة لكرسي السائق ليتحرك نحو الخلف (الشكل 6-20)كفعلٍ تصحيحي.

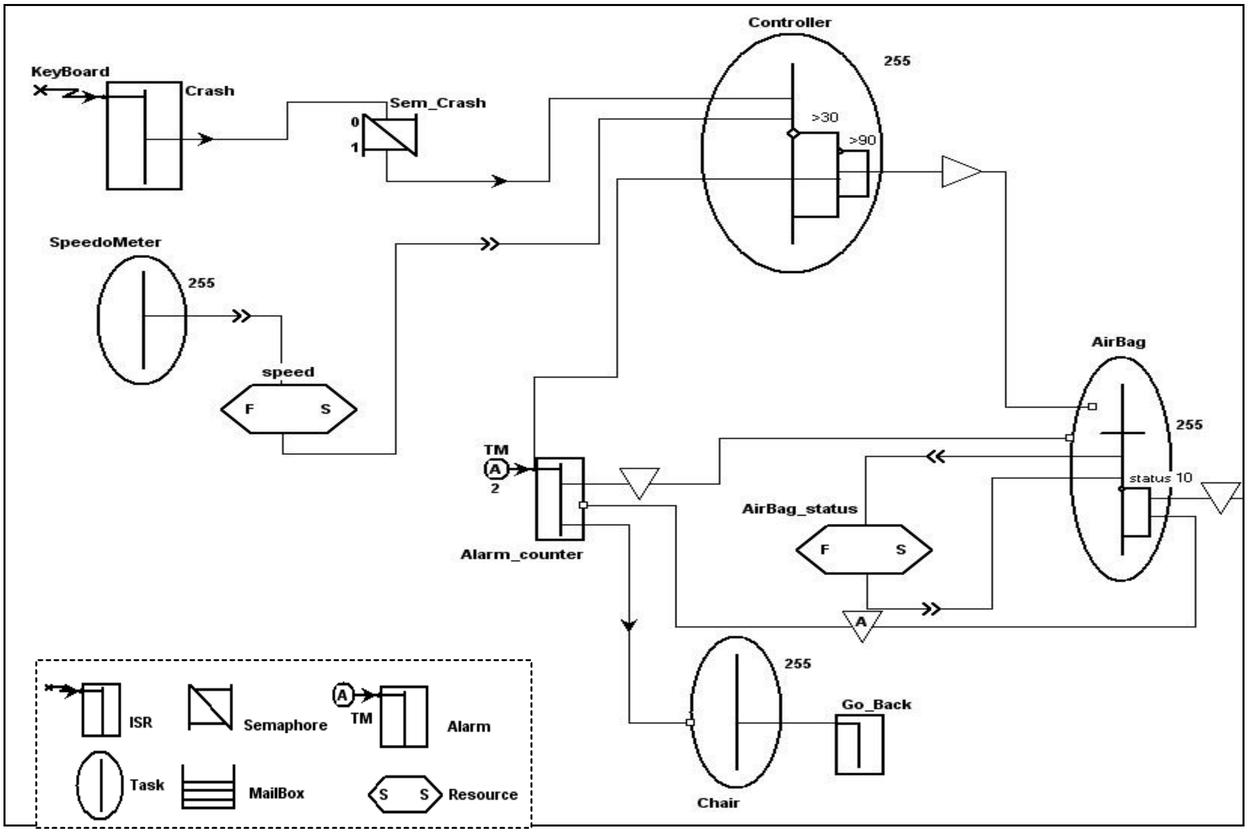
لقد قمنا فعلياً بدراسة هذا النظام وفق الحالات التالية:

- صدمة ← انقضاء ثانيتين ← لم تُفتح الحقيبة ← عودة الكرسي للخلف
- صدمة ← انقضاء زمن أقل ← فُتحت الحقيبة ← لا يرجع الكرسي إلى الخلف
- صدمة بسرعة منخفضة ← لن تحدث أي استجابة




**الشكل(6-20): مخطط تدفق معطيات نظام الحقيبة الهوائية**

ممكن توضيح هذا النظام أيضاً وفق الشكل(6-21) المرسوم ببرنامج LACATRE:




**الشكل(6-21): مخطط تفصيلي لعمل النظام**

سنقوم الآن بكتابة البرنامج المُعبّر عن نظام الحقيبة الهوائية (الشكل 6-22) والقيود الزمنية المفروضة عليه وفق لغة C/VxWorks. تم مقابلة كل عملية مطلوبة من النظام بمهمة تتضمن عدداً من التعليمات المؤدية للهدف المطلوب منها وهذه المهام هي (الحقيبة، الكرسي، الصدمة، مهمة تحكمية عامة). لتأمين الاتصال و التزامن بين هذه المهام أثناء عملها قمنا بإضافة سيمافورات ثنائية تُنظّم أداء هذه المهام عند وجود تشارك فيما بينها على المصادر المختلفة المطلوبة للعمل (تحرير أو حجز مصدر معين من مهمة ما يؤثر في أداء المهمة الأخرى). السيمافورات المُعرّفة هي:

- سيمافور الصدمة: يدل تحريره على حدوث الصدمة مما يتيح لمهمة التحكم متابعة عملها.
- سيمافور الكرسي: يتم تحريره عند انتهاء الزمن الأعظمي المُتاح لاستجابة الحقيبة مما يؤدي إلى تفعيل عمل الكرسي.
- سيمافور لانتهاء عملية نفخ البالون: يتحرر لحظة انتهاء نفخ البالون ليسمح بالقيام بفتح الحقيبة من قبل المهمة الخاصة بها.

أما ضبط زمن تنفيذ هذا التطبييق فقد تم من خلال وضع مؤقت زمني خاص بالزمن الفعلي المستغرق لفتح الحقيبة ومؤقت آخر للزمن الأعظمي المتاح لفتحها. ولقد تم التعبير عن حدث التقاط إشارة من الوسط

الخارجي ليُعبّر عن حدوث صدمة بمهمة خاصة (تُعتبر هذه الإشارة بمثابة مقاطعة خارجية للنظام). تتواصل هذه العناصر مع بعضها البعض لتؤدي العمل المسند لها على أكمل وجه من خلال التوابع الخاصة بكل منها.

```
#include <vxworks.h>
#include <vxworks.h>
#include <tasklib.h>
#include <time.h>
#include <stdio.h>
#include <math.h>
#include <semlib.h>
#include <wdLib.h>
#include <string.h>
#include <stdlib.h>
LOCAL int crashid;
LOCAL int controllerid;
LOCAL int airbagid;
LOCAL int chairid;
LOCAL WDOG_ID Alarm_counter;
LOCAL WDOG_ID exc_blowing;
LOCAL SEM_ID sem_crash;
LOCAL SEM_ID sem_chair;
LOCAL SEM_ID endTime;
void crash(void);
void controller(void);
void airbag(void);
void chair(void);
void Observable_Working(void);
void ExecutingTimeAL(void);
int main (void)
{       sem_crash = semBCreate(SEM_Q_FIFO,SEM_EMPTY);
        sem_chair = semBCreate(SEM_Q_FIFO,SEM_EMPTY);
        endTime = semBCreate(SEM_Q_FIFO,SEM_EMPTY);
        Alarm_counter=wdCreate();
        exc_blowing=wdCreate();
        chairid=taskSpawn("task_chair",20,0,200,(FUNCPTR)chair,0,0,0,0,0,0,0,0,0,0);
        airbagid=taskSpawn("task_airbag",20,0,200,(FUNCPTR)airbag,0,0,0,0,0,0,0,0,0,0);
        taskSuspend(airbagid);
        controllerid=taskSpawn("task_controller",20,0,200,(FUNCPTR)controller,0,0,0,0,0,0,0,0,0,0);
        crashid=taskSpawn("task_crash",20,0,200,(FUNCPTR)crash,0,0,0,0,0,0,0,0,0,0);
        printf(" This Program is written to Simulate The AirBag System\n");
        return 0;
}
void crash(void)
{
        if ( getchar())
        {    printf("There is accident, Be Careful \n");
             semGive(sem_crash);
        }
}
void controller(void)
{       int speedcar;
        speedcar=70;
        printf("The Controller now is working \n");
        semTake(sem_crash,WAIT_FOREVER);
        if (speedcar>30)
        {       if (speedcar<90)
```

```
		{	printf("The speed is between 30-90 \n");
			taskResume(airbagid);
			wdStart(Alarm_counter,50,(FUNCPTR)Observable_Working,0);
		}
	}
	else
	{
		printf(" the system will not Open the AirBag, Because the speed is less than 30 \n");
	}
}
void airbag(void)
{
	wdStart(exc_blowing,54,(FUNCPTR)ExecutingTimeAL,0);/*this watch dog represents the execution time*/
	semTake(endTime,WAIT_FOREVER);
	printf(" AirBag is Opened so the chair will not move  \n");
	wdCancel(Alarm_counter);
}
void chair(void)
{
	semTake(sem_chair,WAIT_FOREVER);
	printf(" Don't worry, The Chair has gone Back \n");
	wdCancel(exc_blowing);
}
void Observable_Working(void)
{
	printf(" The time of alarm finished and the AirBag don't opened at suitable period, so the chair will move \n");
	semGive(sem_chair);
}
void ExecutingTimeAL(void)
{
	printf(" The AirBag is finish blowing now \n");
	semGive(endTime);
}
```

**الشكل(6-22): البرنامج وفق لغة C/VxWorks**

يُقابل البرنامج السابق حالة إزاحة الكرسي نحو الخلف نظراً لأن الزمن الأعظمي المُتاح لاستجابة الحقيبة (50 ticks) أقل من الزمن الفعلي المطلوب لفتحها (54 ticks). لدى استخدام مترجمنا لتحويل هذا البرنامج المكوّن من تسعة عناصر للزمن الحقيقي حصلنا على خرج المترجم وهو عبارة عن برنامج بلغة IF يحتوي على نظام مؤلف من (13) عملية (عمليات تُقابل المهام المعرّفة وعمليتان للمؤقتات الزمنية وعمليتان لبرامج الخدمة المقابلة للمؤقتات و٣ سيمافورات ثنائية). ثم تابعنا معالجة هذا البرنامج بأدوات IF لتوليد ملف المحاكاة الشامل المقابل له، ثم طبقنا خوارزميات المعالجة على ملف المحاكاة للتأكّد من صحة القيود الزمنية المفروضة على هذا النظام ولتحقيق ذلك قمنا بمراقبة مايلي:

١. إشارة ورود صدمة.
٢. لحظة الانتهاء من عملية فتح الحقيبة.
٣. عملية إرجاع الكرسي نحو الخلف.

تم دراسة كافة السيناريوهات الممكنة والمتاحة من خلال تكرار تنفيذ هذا التطبيق مع تغيير قيم المتحولات المؤثرة في نتائج التطبيق (الجدول 6-1)و هذه المتحولات هي:

- سرعة السيارة لحظة الاصطدام، يتم تحديدهابرمجياً و فق قيمة المتحول (speed_car).
- الزمن المستغرق من قبل الحقيبة والمطلوب لنفخ البالون،حدّدبرمجياً من خلال زمن تفعيل المؤقت الزمني (exc_blowing) ذي البرنامج (ExecutingTimeAL).
- الزمن الأعظمي المتاح للاستجابة الفعلية للحقيبةفي حال تجاوز ه دون استجابة الحقيبة سيتمتحر يك الكرسي نحو الخلف لضمان حماية السائق، ولقدحدّد ذلكبرمجياً من خلال المؤقت الزمني (Alarm_counter) ذي برنامج الخدمة (Observable_Working)الذي يُعبّر عن الزمن المتاح للاستجابة. هذه المتحولات تتوافق مع ما هو مطلوب للمراقبة.

| توصيف الحالة المدروسة | | خرج البرنامج وفق C/VxWorks | النتائج الفعلية | |
|---|---|---|---|---|
| | | | عدد الحالات | عدد المناقلات |
| **الصدمة خفيفة**<br>Speed_Car =20 (سرعة السيارة أقل من ٣٠)<br>***The expected results: The system will not do anything*** | | ✓ The Controller now is working ***1***<br>✓ There is accident,Be Careful<br>✓ The system will not Open the AirBag, Because the speed is less than 30. | **262465** | **1857522** |
| **الصدمة قوية**<br>(سرعة السيارة بين ٣٠-٩٠)<br>Speed_Car =70 | ***The expected result: The chair will move***<br>**ExecutingTimeAL=54, Observable_Working =50** | ✓ The Controller now is working ***1***<br>✓ There is accident,Be Careful<br>✓ The speed is between 30-90<br>✓ The time of alarm finished and the AirBag don't opened at suitable period, so the chair will move<br>✓ Don't worry, The Chair has gone Back | **262514** | **1857611** |
| | ***The expected result: The Airbag will open***<br>**ExecutingTimeAL=50, Observable_Working =54** | ✓ The Controller now is working ***1***<br>✓ There is accident, Be Careful<br>✓ The speed is between 30-90<br>✓ The AirBag is finish blowing now<br>✓ AirBag is Opened so the chair will not move | **262514** | **1857611** |

**الجدول(6-1): النتائج العملية لتطبيق الحقيبة الهوائية**

### 1.1.3.6 حالة رجوع الكرسي نحو الخلف:

في هذه الحالة تم توصيف زمن فتح الحقيبة بحيث يكون أكبر أو يساوي قيمة الزمن الأعظمي المتاح (Deadline=3)، وتم اختبار ذلك من خلال تحديد قيمة الزمن الأعظمي المتاح للاستجابة ليأخذ قيمة أقل مع ثلاث ثوانٍ (3*18=54) أما الزمن الفعلي المستغرق لفتح الحقيبة فهو يساوي تلك القيمة مما سيؤدي إلى عدم استجابة الحقيبة كما يجب (يمكن أن تحدث هذه الحالة نتيجة وضعية Deadlock) وبالتالي سيعود الكرسي إلى الخلف،وقد تم التعبير عن هذه الحالة وفق مقطع تعليمات IF التالي:

```
process airbagid(1) ; …
state executing_0;
output wdStart(54) to ({exc_blowing}0);
output semBTake(self,-1) to ({endTime}0);
nextstate waiting_0;
endstate; …
```

```
endprocess;
process exc_blowing(1);
var  alarm_delay integer; …
state ready;
  input wdDelete();
     nextstate dormant;
  input wdStart(alarm_delay);
    set alarm_time :=0;
    nextstate armed;
endstate;
state armed;
  input wdCancel();
     reset alarm_time;
     nextstate ready;
 when alarm_time = alarm_delay;
    output execute_routine() to ({exc_blowing_alarm_Routine}0);
    nextstate ready;
  input wdDelete();
     nextstate dormant;
endstate; …
endprocess;
process controllerid(1) ;
state executing_1;
   if ( speedcar > 30 ) then
     if ( speedcar < 90 ) then
          informal "The speed is between 30 - 90 ";
         output taskResume() to ({airbagid}0);
         output wdStart(50) to ({Alarm_counter}0);
endif
else
informal "the system will not Open the AirBag Because the speed is less than 30 ";
endif
nextstate ready_2;
endstate;  …
endprocess;
process Alarm_counter(1);
var  alarm_delay integer;
var alarm_time  clock;
…
state ready;
  input wdDelete();
     nextstate dormant;
  input wdStart(alarm_delay);
    set alarm_time :=0;
    nextstate armed;
endstate;
state armed;
  input wdCancel();
     reset alarm_time;
     nextstate ready;
 when alarm_time = alarm_delay;
    output execute_routine() to ({Alarm_counter_alarm_Routine}0);
    nextstate ready;
  input wdDelete();
     nextstate dormant;
endstate;
endprocess;
```

تم تحديد ملف المراقبة بحيث يحتوي كلمات من العبارات التالية:

- Controller now is **working**
- There is **accident**
- speed is **between**
- chair will **move**
- Chair has **gone** Back

حصلنا نتيجة لمراقبة الكلمات السابقة (ذات نمط الخط الغامق) على الملف الذي يتضمن المناقلات التالية:

des(0,10,10)
(0,"{controllerid}0 The Controller now is working {controllerid}0 !semBTake{{controllerid}0,-1} ",2)
(0,"{crashid}0 There is accident Be Careful ",1)
(1,"{controllerid}0 The Controller now is working {controllerid}0 !semBTake{{controllerid}0,-1} ",3)
(2,"{crashid}0 There is accident Be Careful ",3)
(3,"{controllerid}0 The speed is between 30 - 90 {controllerid}0 !taskResume{} {controllerid}0 !wdStart{50} ",4)
(3,"{controllerid}0 The speed is between 30 - 90 {controllerid}0 !taskResume{} {controllerid}0 !wdStart{50} ",5)
(4,"{Alarm_counter_alarm_Routine}0 ?execute_routine{} {Alarm_counter_alarm_Routine}0 The time of alarm finished and the AirBag don t opened at suitable period so the chair will move {Alarm_counter_alarm_Routine}0 !semGive{{Alarm_counter_alarm_Routine}0} ",6)
(5,"{Alarm_counter_alarm_Routine}0 ?execute_routine{} {Alarm_counter_alarm_Routine}0 The time of alarm finished and the AirBag don t opened at suitable period so the chair will move {Alarm_counter_alarm_Routine}0 !semGive{{Alarm_counter_alarm_Routine}0} ",7)
(6,"{chairid}0 Don t worry The Chair has gone Back {chairid}0 !wdCancel{} ",8)
(7,"{chairid}0 Don t worry The Chair has gone Back {chairid}0 !wdCancel{} ",9)

نلاحظ أن بعض هذه المناقلات يؤدي إلى حالة مخطط غير محدد وبإلغائها برمجياً ينتج لدينا الملف المعدّل الذي يتضمن المناقلات التالية:

des(0,7,7)
(0,"{controllerid}0 The Controller now is working {controllerid}0 !semBTake{{controllerid}0,-1} ",2)
(0,"{crashid}0 There is accident Be Careful ",1)
(1,"{controllerid}0 The Controller now is working {controllerid}0 !semBTake{{controllerid}0,-1} ",3)
(2,"{crashid}0 There is accident Be Careful ",3)
(3,"{controllerid}0 The speed is between 30 - 90 {controllerid}0 !taskResume{} {controllerid}0 !wdStart{50} ",4)
(4,"{Alarm_counter_alarm_Routine}0 ?execute_routine{} {Alarm_counter_alarm_Routine}0 The time of alarm finished and the AirBag don t opened at suitable period so the chair will move {Alarm_counter_alarm_Routine}0 !semGive{{Alarm_counter_alarm_Routine}0} ",5)
(5,"{chairid}0 Don t worry The Chair has gone Back {chairid}0 !wdCancel{} ",6)

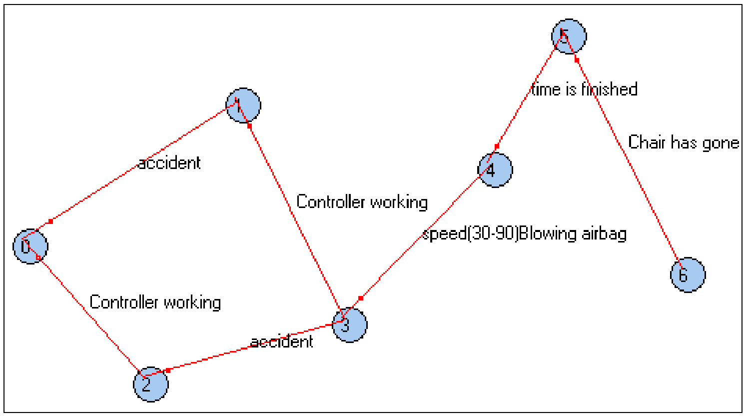
فعلياً يمكننا رسم المخطط المقابل لهذا الملف مباشرة غير أننا لإيضاحه بصورة أفضل قمنا بإعادة تسمية مناقلاته وإعطائها تسميات أكثر تعبيراً عن الحالة المدروسة، وقد حددنا ملف إعادة التسمية بحيث يتضمن المعلومات التالية:

{controllerid}0 The Controller now is working {controllerid}0 !semBTake{{controllerid}0,-1} % Controller working %
{crashid}0 There is accident Be Careful % accident %
{controllerid}0 The speed is between 30 - 90 {controllerid}0 !taskResume{} {controllerid}0 !wdStart{50} % speed(30-90)Blowing airbag %
{Alarm_counter_alarm_Routine}0 ?execute_routine{} {Alarm_counter_alarm_Routine}0 The time of alarm finished and the AirBag don t opened at suitable period so the chair will move {Alarm_counter_alarm_Routine}0 !semGive{{Alarm_counter_alarm_Routine}0} % time is finished %
{chairid}0 Don t worry The Chair has gone Back {chairid}0 !wdCancel{} % Chair has gone %

باستخدام ملف إعادة التسمية السابق لإعادة تسمية مناقلات الملف المعدّل نتج لدينا الملف النهائي التالي:

des(0,7,7)
(0," Controller working ",2)
(0," accident ",1)
(1," Controller working ",3)
(2," accident ",3)
(3," speed(30-90)Blowing airbag ",4)
(4," time is finished ",5)
(5," Chair has gone ",6)

برسم مخطط الأتومات المقابل لهذا الملف نحصل على الشكل التالي (6-23) الذي يوضح كيف أن الكرسي فعلياً عاد نحو الخلف لدى تجاوز القيد الزمني المدروس.




الشكل(6-23): مخطط الأتومات المقابل لحالة تحرك الكرسي

### 2.1.3.6 حالة فتح الحقيبة الهوائية:

تم توصيف زمن فتح الحقيبة الفعلي بحيث يكون أقل من قيمة الـ(Deadline=3)وبناءً عليه سيتم فتح الحقيبة في اللحظة المناسبة ولن يكون هناك حاجة لتحريك الكرسي إطلاقاً، وقد قمنا بمعالجة ملف المحاكاة الشامل المقابل لهذه الحالة ومراقبة العناصر التالية (كلمات من العبارات):

- Controller now is **working**
- There is **accident**
- speed is **between**
- AirBag is **finish** blowing
- AirBag is **Opened**

نتيجة هذه المراقبة حصلنا على ملف المحاكاة الذي يتضمن المناقلات التالية:

des(0,10,10)
(0,"{controllerid}0 The Controller now is working {controllerid}0 !semBTake{{controllerid}0,-1} ",2)
(0,"{crashid}0 There is accident Be Careful ",1)
(1,"{controllerid}0 The Controller now is working {controllerid}0 !semBTake{{controllerid}0,-1} ",3)
(2,"{crashid}0 There is accident Be Careful ",3)
(3,"{controllerid}0 The speed is between 30 - 90 {controllerid}0 !taskResume{} {controllerid}0 !wdStart{54} ",4)
(3,"{controllerid}0 The speed is between 30 - 90 {controllerid}0 !taskResume{} {controllerid}0 !wdStart{54} ",5)
(4,"{exc_blowing_alarm_Routine}0 ?execute_routine{} {exc_blowing_alarm_Routine}0 The AirBag is finish blowing now {exc_blowing_alarm_Routine}0 !semGive{{exc_blowing_alarm_Routine}0} ",6)

(5,"{exc_blowing_alarm_Routine}0 ?execute_routine{} {exc_blowing_alarm_Routine}0 The AirBag is finish blowing now {exc_blowing_alarm_Routine}0 !semGive{{exc_blowing_alarm_Routine}0} ",7)
(6,"{airbagid}0 AirBag is Opened so the chair will not move {airbagid}0 !wdCancel{} ",8)
(7,"{airbagid}0 AirBag is Opened so the chair will not move {airbagid}0 !wdCancel{} ",9)

نلاحظ كذلك أن هذا الملف يتضمن بعض المناقلات التي تؤدي إلى مخطط غير محدد، وبمعالجتها وإلغائها وفق الشكل المناسب (تم شرح أسلوب المعالجة في الفصل السابق) نحصل على المناقلات المعدّلة التالية:

des(0,7,7)
(0,"{controllerid}0 The Controller now is working {controllerid}0 !semBTake{{controllerid}0,-1} ",2)
(0,"{crashid}0 There is accident Be Careful ",1)
(1,"{controllerid}0 The Controller now is working {controllerid}0 !semBTake{{controllerid}0,-1} ",3)
(2,"{crashid}0 There is accident Be Careful ",3)
(3,"{controllerid}0 The speed is between 30 - 90 {controllerid}0 !taskResume{} {controllerid}0 !wdStart{54} ",4)
(4,"{exc_blowing_alarm_Routine}0 ?execute_routine{} {exc_blowing_alarm_Routine}0 The AirBag is finish blowing now {exc_blowing_alarm_Routine}0 !semGive{{exc_blowing_alarm_Routine}0} ",5)
(5,"{airbagid}0 AirBag is Opened so the chair will not move {airbagid}0 !wdCancel{} ",6)

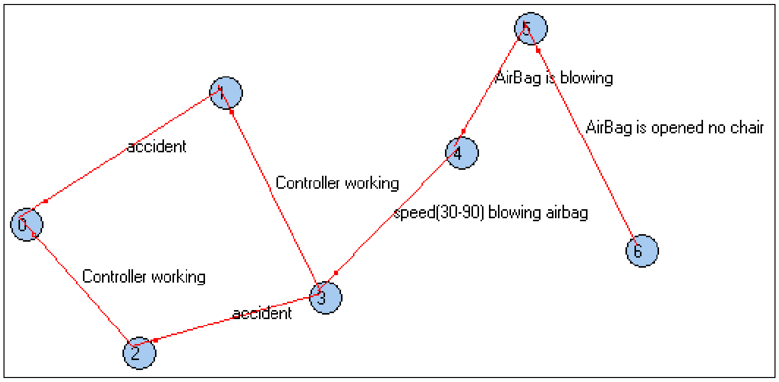
قمنا بتحديد ملف إعادة تسمية المناقلات بحيث يتضمن المعلومات التالية:

{controllerid}0 The Controller now is working {controllerid}0 !semBTake{{controllerid}0,-1} % Controller working %
{crashid}0 There is accident Be Careful % accident %
{controllerid}0 The speed is between 30 - 90 {controllerid}0 !taskResume{} {controllerid}0 !wdStart{54} % speed(30-90) blowing airbag %
{exc_blowing_alarm_Routine}0 ?execute_routine{} {exc_blowing_alarm_Routine}0 The AirBag is finish blowing now {exc_blowing_alarm_Routine}0 !semGive{{exc_blowing_alarm_Routine}0} % AirBag is blowing %
{airbagid}0 AirBag is Opened so the chair will not move {airbagid}0 !wdCancel{} % AirBag is opened no chair %

بالنتيجة يصبح الملف النهائي بعد إعادة تسمية مناقلاته وفق الشكل التالي:

des(0,7,7)
(0," Controller working ",2)
(0," accident ",1)
(1," Controller working ",3)
(2," accident ",3)
(3," speed(30-90) blowing airbag ",4)
(4," AirBag is blowing ",5)
(5," AirBag is opened no chair ",6)

المخطط المقابل (الشكل 6-24) لهذا الملف يوضح كيف أن الحقيبة الهوائية تم فتحها بشكل صحيح وفق هذه الحالة، وهذا ما يُعبّر عن احترام القيد الزمني.




**الشكل(6-24): مخطط الأتومات المقابل لحالة فتح الحقيبة الهوائية**

### 3.1.3.6 حالة السرعة بطيئة لا تتطلب أي استجابة:

في هذه الحالة تم إعطاء السرعة قيمة أصغر من قيمة السرعة الحدية التي تقتضي حدوث استجابة من قبل الحقيبة الهوائية وبعد الحصول على ملف المحاكاة الشامل قمنا بتحديد ملف المراقبة الخاص بهذه الحالة بحيث يحتوي كلمات من العبارات التالية:

- The Controller now is **working**
- There is **accident** Be Careful
- the system will not Open the AirBag Because the speed is **less** than 30

نتيجة لمراقبة الكلمات المطلوبة التي توصف على عمل النظام وفق وضعيات خاصة تتوافق مع الحالة المدروسة، حصلنا على ملف المحاكاة الذي يتضمن المناقلات التالية:

```
des(0,6,6)
(0,"{controllerid}0 The Controller now is working  {controllerid}0 !semBTake{{controllerid}0,-1} ",2)
(0,"{crashid}0 There is accident Be Careful  ",1)
(1,"{controllerid}0 The Controller now is working  {controllerid}0 !semBTake{{controllerid}0,-1} ",3)
(2,"{crashid}0 There is accident Be Careful  ",3)
(3,"{controllerid}0 the system will not Open the AirBag Because the speed is less than 30  ",4)
(3,"{controllerid}0 the system will not Open the AirBag Because the speed is less than 30  ",5)
```

نلاحظ أن بعض هذه المناقلات يؤدي إلى حالة مخطط غير محدد وهذا يتطلب منا إزالة حالة الغموض ومعالجتها، نتيجة هذه المعالجة نحصل علىالملف المعدّل التالي:

```
des(0,5,5)
(0,"{controllerid}0 The Controller now is working  {controllerid}0 !semBTake{{controllerid}0,-1} ",2)
(0,"{crashid}0 There is accident Be Careful  ",1)
(1,"{controllerid}0 The Controller now is working  {controllerid}0 !semBTake{{controllerid}0,-1} ",3)
(2,"{crashid}0 There is accident Be Careful  ",3)
(3,"{controllerid}0 the system will not Open the AirBag Because the speed is less than 30  ",4)
```

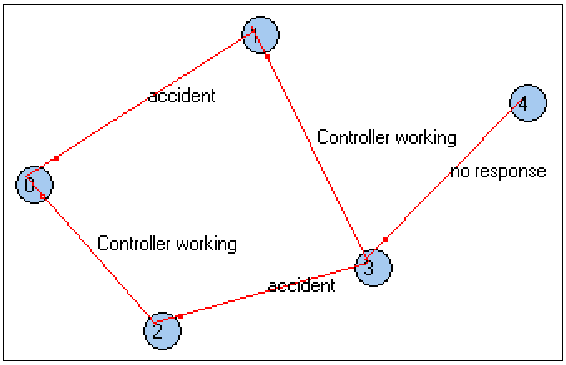
أما ملف إعادة التسمية فقد تم تحديده وفق التالي:

```
{controllerid}0 The Controller now is working  {controllerid}0 !semBTake{{controllerid}0,-1} % Controller working %
{crashid}0 There is accident Be Careful  % accident %
{controllerid}0 the system will not Open the AirBag Because the speed is less than 30  % no response %
```

وبإعادة تسمية المناقلات وفق ما هو مطلوب نحصل على الملف النهائي الذي يتضمن المناقلات التالية:

```
des(0,5,5)
(0," Controller working ",2)
(0," accident ",1)
(1," Controller working ",3)
(2," accident ",3)
(3," no response ",4)
```

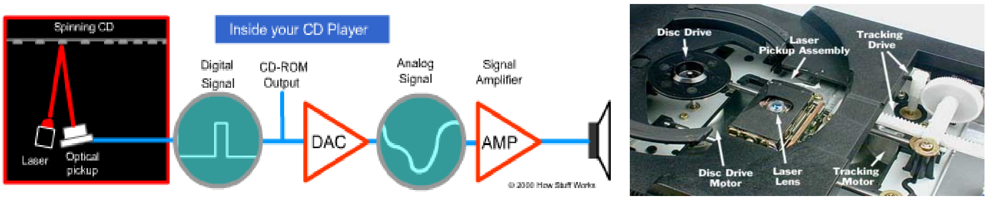
يؤكد المخطط المقابل (الشكل 6-25) لهذا الملف عدم حدوث أي استجابة من قبل النظام عندما تكون قيمة السرعة لحظة الصدمة أقل من القيمة الحدية المطلوبة.




الشكل(6-25): مخطط الأتوماتالمقابل لحالة السر عة أقل من ٣٠

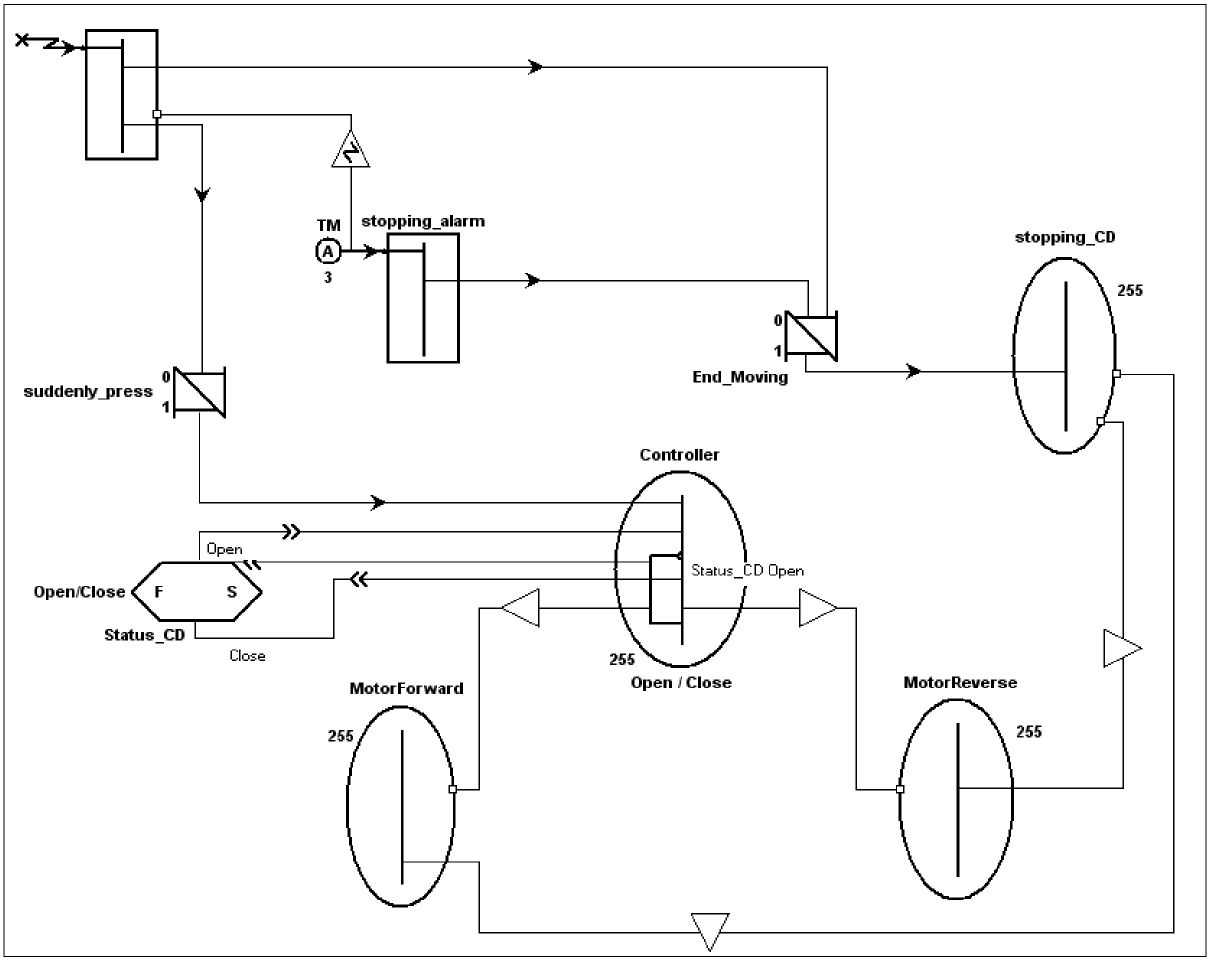
### 2.3.6 تطبيق قارئ السيدي:

سندرس حالياً مثالاً آخر عن نظام زمن حقيقي وهو دراسة عمل قارئ السيديات CD Reader (الشكل 6-26) وما يتطلبه من دقة في زمن المعالجة التي يجب أن تبديها السواقة الليزرية أثناء عملية الفتح أو الإغلاق نتيجة الاستجابة لحدث الضغطالمطبّق على مفتاحها الخارجي. كما نعلم تتحدد استجابة قارئ السيديات (الشكل 6-27) وفق التالي: في بداية الأمر يكون قارئ السيدي مغلقاً أي لا يحتوي أي سيدي ويمكن للمستخدم القيام بالضغط على زر فتح/إغلاق السيدي لينفتح وذلك بتشغيل محركمُعد لهذه الغاية، ومن ثم يمكن لاحقاً إغلاقه وفق الحاجة، نلاحظ أن عملية الفتح أو الإغلاق تأخذ زمناً قدرثلاث ثوانٍ اعتباراً من لحظة الضغط، من ناحية أخرى يمكن للمستخدم في أي لحظةو أثناء حركة السواقة القيام بعملية الضغط على زر الفتح/إغلاق لمرة أخرى مما يؤدي إلى عكس العملية السابقة وتشغيل المحرك بالاتجاه المعاكس ويزمن يساوي الزمن المنقضي بالعملية الأصلية.




الشكل(6-26): عمل قارئ السيدي

على سبيل المثال بفرض أنه تم الضغط على الزر وكانت السيدي مفتوحة فإن المحرك يدور باتجاه عملية الإغلاق ولمدة ثلاثثوانٍ ، بفرض أنه انقضى ثانيتان على دور المحرك بهذا الاتجاه وتم الضغط على الزر ثانية عندها سيؤدي ذلك إلى تشغيل المحرك بالاتجاه المعاكس ولمدة ثانيتين فقط وهكذا، يكون عادة هذا السيناريو متاحاً أيضاً أثناء عملية الفتح.




**الشكل(6-27): مخطط تفصيلي لعمل الـ CD Reader**

تم كتابة البرنامج المقابل لهذا المثال وفق لغة C/VxWorks كالتالي (الشكل 6-28):

```
#include <vxworks.h>
#include<tasklib.h>
#include<time.h>
#include<stdio.h>
#include<math.h>
#include <semlib.h>
#include <wdLib.h>
#include <string.h>
#include <stdlib.h>
LOCAL int pressbuttonid;
LOCAL int controllerid;
LOCAL int motorreverseid;
LOCAL int motorforwardid;
LOCAL int stoppingcdid;
LOCAL WDOG_ID stopping_alarm;
LOCAL SEM_ID end_moving;
LOCAL SEM_ID suddenly_press;
void pressCD(void);
void controller(void);
void motorreverse(void);
void motorforward(void);
```

```
void stoppingcd(void);
void stopMoving_Al(void);
int main (void)
{
        end_moving = semBCreate(SEM_Q_FIFO,SEM_EMPTY);
        suddenly_press = semBCreate(SEM_Q_FIFO,SEM_EMPTY);
        stopping_alarm=wdCreate();
        pressbuttonid=taskSpawn("Pressing_Button",20,0,200,(FUNCPTR)pressCD,0,0,0,0,0,0,0,0,0,0);
        controllerid=taskSpawn("task_controller",20,0,200,(FUNCPTR)controller,0,0,0,0,0,0,0,0,0,0);
        taskSuspend(controllerid);
        motorreverseid=taskSpawn("Motor_Reverse",20,0,200,(FUNCPTR)motorreverse,0,0,0,0,0,0,0,0,0,0);
        taskSuspend(motorreverseid);
        motorforwardid=taskSpawn("Motor_Forward",20,0,200,(FUNCPTR)motorforward,0,0,0,0,0,0,0,0,0,0);
        taskSuspend(motorforwardid);
        stoppingcdid=taskSpawn("StoppingCD",20,0,200,(FUNCPTR)stoppingcd,0,0,0,0,0,0,0,0,0,0);
        printf(" This Program simulates the CD Reader \n");
        return 0;
}
void pressCD(void)
{
        if ( getchar())/* the first press of button*/
        {
                printf("SomeOne press the button to open the cd_Reader \n");
                wdStart(stopping_alarm,54,(FUNCPTR)stopMoving_Al,0); /*watchDog represents time (3) sec*/
                taskResume(controllerid);
        }
}
void stopMoving_Al(void)
{
        printf("The alarmWorking is finished,the stopping Motor activate \n");
        semGive(end_moving);
}
void stoppingcd(void)
{       semTake(end_moving,WAIT_FOREVER);
        printf("The CD is stopped \n");
}
void controller(void)
{
        int statusCD;
        statusCD=0;
        if ( getchar()) /* the second press of button*/
        {
            wdCancel(stopping_alarm);
            semGive(suddenly_press);
             printf("The stopping_alarm is canceled,The Button is pressed suddenly \n");
            if(statusCD==0)
            {
                printf("The CD Reader is opening now, so it will go back \n");
                taskResume(motorreverseid);
                statusCD=1;
            }
            else
            {
                printf("The CD Reader is closing now, it will go forward \n");
                taskResume(motorforwardid);
                statusCD=0;
            }
```

```
        }
}
void motorreverse(void)
{       semTake(suddenly_press,WAIT_FOREVER);
        printf("The motor reversed \n");
        semGive(end_moving);
}
void motorforward(void)
{       semTake(suddenly_press,WAIT_FOREVER);
        printf("The motor moved forward \n");
        semGive(end_moving);
}
```

**الشكل(6-28): البرنامج وفق لغة C/VxWorks**

في الحالة العامة لدى تطبيق هذا البرنامج ضمن نظام VxWorks و القيام بالضغط لمرة واحدة فقط على زر السواقة ليتم فتحها (والذي يُ كافئ برمجياً التقاط إشارة من الوسط الخارجي من خلال النقر على أي مفتاح من لوحة المفاتيح) عندها فإن السواقة ستتحرك نحو الأمام إلى أن ينتهي الزمن المخصص لها (ثلاث ثوانٍ ) ومنه يُفعّل برمجياً المؤقت الزمني ليعطي إشارة للمحرك لإيقاف السواقة، نتيجة لهذه الحالة حصلنا على الخرج التالي:

- *__1__* SomeOne press the button to open the cd_Reader
- The alarmWorking is finished,the stopping Motor activate
- The CD is stopped

ولدى تكرار عملية الضغط على زر السواقة وهي في وضعية حركة فإنها ستستجيب لذلك بالتحرك وفق الاتجاه المعاكس لجهة حركتها لحظة الضغط، وقد تحكمنا برمجياً بحالة السواقة لحظة ضغط مفتاحها للمرة الثانية من خلال تعديل قيمة المتحول (statusCD) المعبّر عن جهة الحركة والذي يأخذ إحدى قيمتين: (0) للدلالة على السواقة المتجهة نحو الأمام لتُفتح و(1) للدلالة على السواقة التي تتحرك نحو الخلف لتُغلق. باستخدام مترجمنا لتحويل هذا البرنامج (الشكل 6-28) المكوّن من عناصر الزمن الحقيقي الثمانية (خمس مهام تم إنشاؤها وتفعيلها ضمن التابع الرئيسي main كما تم التصريح عن سيمافورين إضافة إلى مؤقت زمني) حصلنا على الشيفرة البرمجية المقابلة على شكل برنامج وفق لغة IF وهو عبارة عن نظام يتضمن (١١) عملية تُقابل العناصر السابقة وهي:

- سبع عمليات تُقابل المهام المطلوبة لعمل النظام: getchar، controllerid، pressbuttonid، motorforwardid، main ، stoppingcdid، motorreverseid.
- عمليتان تُقابل السيمافورات المعرّفة: suddenly_press،end_moving .
- عمليتان لضبط عمل المؤقت الزمني إحداهما تُقابل المؤقت والأخرى برنامج الخدمة المرفق به: stopping_alarm_alarm_Routine، stopping_alarm .

ولدى معالجة البرنامج الناتج (خرج مترجمنا) باستخدام أدوات IF سابقة الذكر حصلنا على ملف المحاكاة الشامل الذي يتضمن عدداً من الحالات والمناقلات يختلف وفق السيناريو المدروس (الجدول 6-2).

| توصيف الحالة المدروسة | خرج البرنامج وفق C/VxWorks | النتائج الفعلية | |
|---|---|---|---|
| | | عدد الحالات | عدد المناقلات |
| **السواقة تتحرك نحو الأمام** ليتم فتحها وتم ضغط الزر ثانية يؤدي ذلك إلى عودة السواقة بالاتجاه المعاكس ليتم إغلاقها. السواقة مغلقة: **statusCD=0** | ✓ ***1*** SomeOne press the button to open the cd_Reader<br>✓ ***1*** The stopping_alarm is canceled,The Button is pressed suddenly<br>✓ The CD Reader is opening now, so it will go back<br>✓ The motor reversed<br>✓ The CD is stopped | 826032 | 5667715 |
| **السواقة تتحرك نحو الخلف** بجهة الإغلاق وتم ضغط الزر ثانية يؤدي ذلك إلى حركة السواقة بالاتجاه المعاكس وفتحها من جديد. السواقة مفتوحة: **statusCD=1** | ✓ ***1*** SomeOne press the button to open the cd_Reader<br>✓ ***1*** The stopping_alarm is canceled,The Button is pressed suddenly<br>✓ The CD Reader is closing now, it will go forward<br>✓ The motor moved forward<br>✓ The CD is stopped | 826004 | 5667677 |

**الجدول(6-2): النتائج العملية لتطبيق قارئ السيدي**

انطلاقاً من اللحظة الأولى لتفعيل العمل، التي تمت من خلال ضغط المفتاح الخارجي للسواقة (التقاط إشارة خارجية)و هي مغلقة ليتم فتحها، فإننا إذا قمنا بضغط المفتاح مرة ثانية سنكون أمام إحدى حالتي المعالجة التاليتين:

### 1.2.3.6 السواقة تتجه نحو الأمام:

تم توصيف هذه الحالة على أن حركة السواقة لحظة القيام بالضغط على مفتاحها الخارجي للمرة الثانية كانت باتجاه الأمام (أي يتم فتحها) مما أدى إلى تحركها بالاتجاه المعاكس واتجاهها نحو الخلف ليتم إغلاقها من جديد، وللتأكد من أن النتائج النهائية تتوافق مع ذلك تم معالجة ملف المحاكاة الشامل بحيث قمنا بمراقبة الإشارات التالية:

- !wdStart: إرسال إشارة لتفعيل المؤقت الزمني الذي يضبط زمن توقف حركة السواقة.
- !wdCancel: إرسال إشارة تجاهل للمؤقت نتيجة الضغط على المفتاح للمرة الثانية بحيث لن تتوقف السواقة اعتباراً من لحظة الضغط الأولى وا ٕنما ستتغير استجابتها وحركتها.
- !semGive{{motorreverseid}: إرسال إشارة تحرير السيمافور من قبل العملية motorreverseid ليسمح بتحر ّك السواقة نحو الخلف.
- stopped: إحدى كلمات العبارة "The CD is stopped" التي تدل على توقف حركة السواقة بعد الاستجابة المطلوبة. نتيجة هذه المراقبة حصلنا على الملف الذي يتضمن مايلي:

```
des(0,9,10)
(0,"{controllerid}0 !wdCancel{} {controllerid}0 The stopping_alarm is canceled The Button is pressed suddenly {controllerid}0 The CD Reader is opening now so it will go back {controllerid}0 !taskResume{} ",1)
(0,"{pressbuttonid}0 SomeOne press the button to open the cd_Reader {pressbuttonid}0 !wdStart{54} {pressbuttonid}0 !taskResume{} ",2)
(1,"{pressbuttonid}0 SomeOne press the button to open the cd_Reader {pressbuttonid}0 !wdStart{54} {pressbuttonid}0 !taskResume{} ",3)
```

(2,"{controllerid}0 !wdCancel{} {controllerid}0 The stopping_alarm is canceled The Button is pressed suddenly {controllerid}0 The CD Reader is opening now so it will go back {controllerid}0 !taskResume{} ",4)
(2,"{controllerid}0 !wdCancel{} {controllerid}0 The stopping_alarm is canceled The Button is pressed suddenly {controllerid}0 The CD Reader is opening now so it will go back {controllerid}0 !taskResume{} ",5)
(3,"{motorreverseid}0 The motor reversed {motorreverseid}0 !semGive{{motorreverseid}0} ",6)
(4,"{motorreverseid}0 The motor reversed {motorreverseid}0 !semGive{{motorreverseid}0} ",7)
(6,"{stoppingcdid}0 The CD is stopped ",8)
(7,"{stoppingcdid}0 The CD is stopped ",9)

لدى معالجة هذه المناقلات (تحسينها لإزالة حالة الغموض من المناقلتين المميزتين السابقتين) حصلنا على المناقلات المعدّلة التالية:

des(0,8,9)
(0,"{controllerid}0 !wdCancel{} {controllerid}0 The stopping_alarm is canceled The Button is pressed suddenly {controllerid}0 The CD Reader is opening now so it will go back {controllerid}0 !taskResume{} ",1)
(0,"{pressbuttonid}0 SomeOne press the button to open the cd_Reader {pressbuttonid}0 !wdStart{54} {pressbuttonid}0 !taskResume{} ",2)
(1,"{pressbuttonid}0 SomeOne press the button to open the cd_Reader {pressbuttonid}0 !wdStart{54} {pressbuttonid}0 !taskResume{} ",3)
(2,"{controllerid}0 !wdCancel{} {controllerid}0 The stopping_alarm is canceled The Button is pressed suddenly {controllerid}0 The CD Reader is opening now so it will go back {controllerid}0 !taskResume{} ",4)
(3,"{motorreverseid}0 The motor reversed {motorreverseid}0 !semGive{{motorreverseid}0} ",5)
(4,"{motorreverseid}0 The motor reversed {motorreverseid}0 !semGive{{motorreverseid}0} ",6)
(5,"{stoppingcdid}0 The CD is stopped ",7)
(6,"{stoppingcdid}0 The CD is stopped ",8)

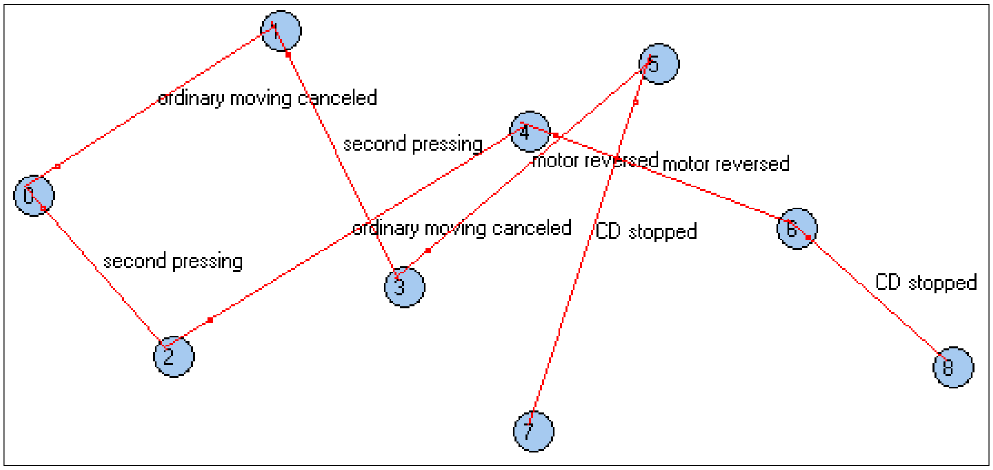
أما ملف إعادة التسمية فقد تم تحديده وفق التالي:

{controllerid}0 !wdCancel{} {controllerid}0 The stopping_alarm is canceled The Button is pressed suddenly {controllerid}0 The CD Reader is opening now so it will go back {controllerid}0 !taskResume{} % ordinary moving canceled %
{pressbuttonid}0 SomeOne press the button to open the cd_Reader {pressbuttonid}0 !wdStart{54} {pressbuttonid}0 !taskResume{} % second pressing %
{motorreverseid}0 The motor reversed {motorreverseid}0 !semGive{{motorreverseid}0} % motor reversed %
{stoppingcdid}0 The CD is stopped % CD stopped %

وبإعادة تسمية هذه المناقلات (كما هو مطلوب) حصلنا على الملف النهائي التالي:

des(0,8,9)
(0," ordinary moving canceled ",1)
(0," second pressing ",2)
(1," second pressing ",3)
(2," ordinary moving canceled ",4)
(3," motor reversed ",5)
(4," motor reversed ",6)
(5," CD stopped ",7)
(6," CD stopped ",8)

يُقابل المخطط التالي (الشكل 6-29) الملف النهائي السابق ويوضح كيف أنه قد تم فعلياً الاستجابة لضغطة الزر الثانية لدى تحرك السيدي نحو الأمامحيث تم تغيير جهة حركته بالاتجاه المعاكس وا ٍغلاق السيدي.




الشكل(6-29): المخطط النهائي وفق المحور الأفقي

### 2.2.3.6 السواقة تتجه نحو الخلف:

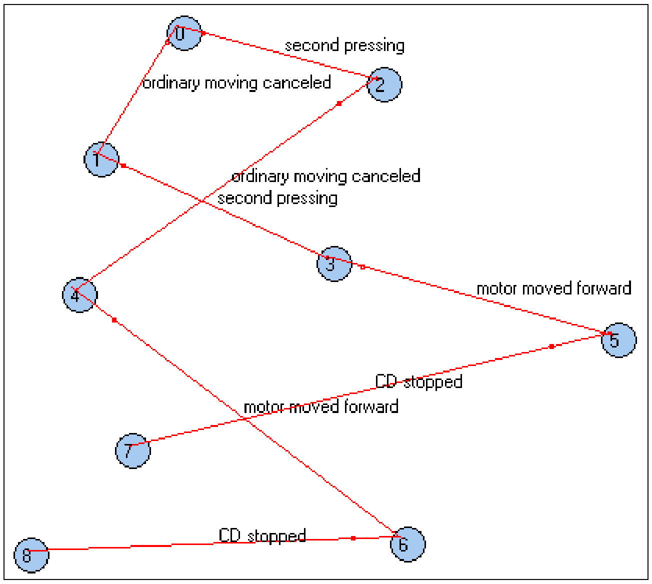
تم توصيف هذه الحالة على أن حركة السواقة لحظة القيام بالضغط على مفتاحها الخارجي للمرة الثانية كانت باتجاه الخلف (أي يتم إغلاقها) مما أدى إلى تحركها بالاتجاه المعاكس واتجاهها نحو الأمام ليتم فتحها من جديد، ولدى معالجة ملف المحاكاة الشامل قمنا بمراقبة العناصر التالية:

(!wdStart{ , !wdCancel{} , stopped , !semGive{{motorforwardid})

وبمناقشة مماثلة للحالة السابقة وبعد تطبيق كافة مراحل المعالجة المطلوبة على ملف المحاكاة الشامل حصلنا على الملف النهائي الذي يتضمن المناقلات التالية:

```
des(0,8,9)
(0," ordinary moving canceled ",1)
(0," second pressing ",2)
(1," second pressing ",3)
(2," ordinary moving canceled ",4)
(3," motor moved forward",5)
(4," motor moved forward",6)
(5," CD stopped ",7)
(6," CD stopped ",8)
```

ولدى رسم المخطط البياني المقابل لهذا الملف نحصل على (الشكل 6-30) الذي يتوافق مع حالة قارئ السيدي المدروسة ويؤكد أن استجابته كانت صحيحة وتتطابق مع السلوك المنتظر.



الشكل(6-30): المخطط النهائي وفق المحور الشاقولي

## 4.6 مناقشة النتائج:

تمت عملية التحقق من صحة نتائج مترجمنا وفق مرحلتين متكاملتين معاً :

- مرحلة التحقق القواعدي من صحة تعليمات لغة IF: التي تتم من خلال إجراء عملية الـ making على البرنامج الناتج عن مترجمنا لإنتاج الملف (ذي الامتداد x.*)الذي يُعطي بتنفيذه ملف المحاكاة الشامل. في حال وجود أي خطأ قواعدي في هذا البرنامج ستظهر عبارة تدل على ذلك ولن يتم الحصول على الملف المنتظر.
- مرحلة التحقق من صحة السلوك المُتّبع في النظام والمقابل للبرنامج المكتوب وفق لغة C/VxWorks: تمت من خلال مراقبة المناقلات الناتجة في ملف المحاكاة الشامل (ذي الامتداد aut.*) والتأكد من شموليتها لكافة الحالات المتوقعة والمطابقة لنتيجة تنفيذ برامج الدخل ضمن نظام VxWorks. أخيراً قمنا بمقارنة يدوية للنتائج التي تم الحصول عليها وتحديد مدى توافقها مع السلوك المتوقع من النظام المدروس.
- أما عملية التحقق من صحة المخطط النهائي فهي إجراء روتيني يعتمد على شمولية خوارزميات المعالجة (المراقبة، إزالة الغموض، إعادة التسمية، إعادة الترقيم) التي ذكرناها في الفصل السابق وعلى صحة نتائجها المعطاة وقد تم اختبارها وفق عدد من المخططات المختلفة للتأكد من دقة مناقلات المخطط النهائي الناتج عن تطبيقها.
- تأكدنا أيضاً خلال عملية التحقق من سلوكية النظام من فعالية تعليمات لغة IF المُقابلة لعناصر الزمن الحقيقي الموصّفة لبرنامج الدخل المترجم. حيث وجدنا أن عدد الحالات والمناقلات في ملف المحاكاة الشامل المقابل للنظام المدروس المكتوب وفق لغة IF(خرج المترجم) يتوقف على عدد عناصر الزمن

الحقيقـي المعر ّ فـة ضـمن برنـامج C/VxWorks وتوابعهـا المسـتدعاة ويتعلـق بطبيعـة المعالجـة المقترحـة للعنصر. يوضح ذلك الجدول التالي (6-3):

| التطبيق | عدد العناصر الموجودة في البرنامج | النتائج الفعلية | |
|---|---|---|---|
| | | عدد الحالات | عدد المناقلات |
| برنامج المهمة | البرنامج يحتوي مهمتين فقط | ١٧ | 17 |
| برنامج المؤقت الزمني | مهمتين + مؤقت زمني | ٤٤ | ٨٦ |
| برنامج المؤقت الزمني المعدّل | مهمتين + مؤقت زمني + سيمافور ثنائي | ٧٧ | ١٧٣ |
| برنامج علبة البريد | مهمتين + علبة بريد | 56 | 94 |
| برنامج السيمافور الثنائي | ثلاث مهام + سيمافور ثنائي | ١٥٧ | ٣٦٨ |
| برنامج السيمافور الثنائي المعدّل | ثلاث مهام + سيمافور ثنائي+استدعاء تابعين | ٣١٩ | ٨٣٣ |

**الجدول(6-3): مقارنة نتائج التطبيقات**

إن صحة نتائج بحثنا هذا أمر حتمي لا يحتمل الارتياب نظراً لكون البرامج الناتجة عن المترجم إما أن تكون صحيحة فتؤدي السلوك المتوقع بدقةو الهدف البرمجي المطلوب أو تكون خاطئة فلا تعطي أي نتائج ولن نحصل عندها علىأيٍ من ملفات المعالجة اللاحقة، وكذلك الأمر بالنسبة للمعالجات الأخرى المتضمنة في البحث.

## 5.6 الخاتمة:

تأكدنا خلال هذا الفصل من صحة نتائج مترجمنا من خلال استخدامه في تحويل ومقابلة التعليمات لعدد من التطبيقات العملية البسيطة،أخيراً تم اختبار مختلف مراحل البحث والتأكد من تحقيقها المطلوب ودقتها من خلال دراسة نظام زمن حقيقي متكامل يتناول عمل الحقيبة الهوائية المرفقة مع سيارة النقلو اختبار السيناريوهات المختلفة الممكنة لعملها، ومن ثم قمنا بدراسة التجاوزات الممكن حدوثها خلال عمل هذا النظام ووضعيات الاستجابة المختلفة التي قدتُبديها الحقيبة لحماية السائق من الاصطدام بهدف التأكد من صحة التحقق من القيود الزمنية، كما اختبرنا ذلك من خلال تطبيق عملي آخر وهو قارئ السيدي وكيفية ضبط استجاباته ضمن نظام الزمن الحقيقي.

بهذا نكون قد قمنا بتحقيق الهدف العلمي المنشود من هذا البحث وأصبح بإمكاننا اعتماد مترجمه كأداة برمجية جديدة موثوقة متاحة للاستخدام ضمن أنظمة الزمن الحقيقي المختلفة.

# ٧

# الفصل السابع

# الملخص ومناقشة النتائج

1.7 الملخص.

2.7 النتائج.

3.7 آفاق مستقبلية للتطوير.

# الملخص ومناقشة النتائج

## 1.7 الملخص:

لقد توصلنا خلال هذا البحث إلى إيجاد طريقة علمية موثوقة ذات جودة عالية يسهل تطبيقها ضمن أنظمة الزمن الحقيقي للتحقق المباشر من صحة القيود الزمنية التي قد تُفرض عليها.

رغم صعوبة التعامل مع هذه الأنظمة والتحكم بها نظراً لدرجة تعقيدها واستحالة قيادتها سابقاً (قبل تقديم بحثنا هذا)، إلا أن عملية التأكد من دقة تنفيذها والتحقق منها أصبحت حاجةً ملحّة يُمليها علينا عالمنا الحالي المعاصر الذي غدت فيه الإلكترونيات والبرمجيات الحاسوبية هي المسيطر الفعلي على كافة تفاصيل حياتنا اليومية، فالتحقق من صحة البرمجيات التي ننتجها في هذا المجال قبل طرحها في سوق العمل أمر يبعث في نفس المبرمج الثقة والطمأنينة بأن ما بذله من مجهود ووقت سيكون له أثر إيجابي في حياة الآخرين يُحقق الهدف المنشود منه ولن يضيع سدى مسبباً الدمار للوسط المُستخدم ضمنه، كما ستنتهي حالة القلق اتجاه الأضرار الجسيمة التي قد تنجم عن أي حالة معالجة شاذة للنظام لم تؤخذ بعين الاعتبار من قبل المبرمج.

يُعالج هذا البحث موضوعاً مُميّزاً وفريداً من نوعه فهو أول من تناول فكرة التحقق من صحة أنظمة الزمن الحقيقي من خلال بناء مترجم خاص لبيئتين برمجيتين مستقلتين كل منهما تهتم بالزمن الحقيقي وتتعامل معه وفق أسلوبها، فدخل مترجمنا هو لغة C/VxWorks التي تُتيح كتابة تطبيقات الزمن الحقيقي أما خرجه فيكون بلغة IF التي تُتيح الأدوات المساعدة في التحقق من صحة القيود الزمنية للزمن الحقيقي والتي لم نجد لها أي مرجع علمي باللغة العربية أو أي دراسة سابقة في هذا المجال.

كما أننا لم نكتفِ خلال بحثنا ببناء هذا المترجم وإنما تابعنا عملنا خلال عدد من خطوات المعالجة المختلفة التي تضمنت حصولنا على مخطط المحاكاة الشامل للتنفيذ ومن ثم طبقنا عليه معايير مراقبة وفلترة خاصة تسمح لنا باكتشاف التجاوزات الزمنية للقيود المدروسة. أخيراً أتحنا رسم الأتومات الزمنية المُعبّر عن المسارات المقابلة للقيود الزمنية المدروسة وفق أداة بيانية تعرضه بأسلوب واضح ومتقن. اعتمدنا في هذا البحث على بيئات وأدوات برمجية مختلفة ساهمت معاً في تحقيق هدفنا المنشود وبروزه لعالم النور.

## 2.7 النتائج:

- يتميز المترجم الناتج عن هذا البحث بوظيفة برمجية خاصة تهدف إلى تحويل تعليمات برامج الزمن الحقيقي المكتوبة بلغة C/VxWorks إلى لغة IF للتحقق المباشر من صحة قيودها الزمنية. وقد تم ذلك بإتباع عدد من خطوات التحليل والتصميم والتطوير البرمجي التي أدت بمجملها إلى إنتاج مترجمنا. المترجم المصمم موثوق ذو جودة برمجية عالية يمتاز بقابلية التعديل والتطوير بسهولة نظراً لاعتماد أسلوب البرمجة غرضية التوجه أثناء برمجة تعليماته.
- تم وفق هذا البحث تقديم دراسة عمليةمفصّلة عن كيفية معالجة البرامج المكتوبة وفق لغة IF، التي تُمثل ملفات الخرج الناتجة عن مترجمنا، وذلك باستخدام أدوات برمجية متاحة من قبل لغة IF تسمح بالحصول على مخطط المحاكاة الشامل المتضمن لكافة حالات التنفيذ الممكنة المقابلة للنظام الموصّف.
- إتاحة إمكانية التحقق من صحة القيود الزمنية المفروضة على أنظمة الزمن الحقيقي من خلال معالجة مخطط الحالات الشامل بطريقة علمية مميزة وفق عدد من الخوارزميات الخاصة المؤدية لهذا الهدف، التي تأخذ دخلها على شكل ملفات ذات الامتداد (aut.) وتعالجها بأسلوب يحافظ فقط على المناقلات المقابلة للقيد الزمني المطلوب التحقق من صحته. علماً أنه كان من المفترض استخدام تقنية CADP الجاهزة للقيام بهذه المعالجة غير أن عدم تأمينها كونها برمجية غير مفتوحة المصدر أضاف مهام جديدة على هذا البحث ساعدتنا بمجملها في عملية التحقق من صحة النتائج.
- تقديم أداة رسومية بسيطة تأخذ معطياتها من مخطط المحاكاة النهائي وتعالجها وتعرضهابيانياً بأسلوب واضح ومميز.
- نشرنا خطوات بناء المترجم والتعليمات المتقابلة بين اللغتين ضمن ورقة عمل في مجلة بحوث جامعة حلب – سلسلة العلوم الهندسية.

## 3.7 آفاق مستقبلية للتطوير:

- تم خلال هذا البحث بناء مترجم يقوم بتحليل برنامج الدخل من الناحية اللفظية والتركيبية دون التحقق الشامل من صحة المعنى الدلالي لتعليماته. يمكننا متابعة تطوير بحثنا ضمن هذا الاتجاه بحيث نقوم بإنجاز مرحلة التحليل هذه بشكل أكثر شمولية يضمن لنا تنبيه المبرمج للأخطاء الخاصة بهذا النمط من التحليل قبل تحويل البرنامج المترجم إلى تعليمات لغة IF المقابلة. على سبيل المثال يجب مراعاة التالي:
  - عدم التعامل مع أي عنصر برمجي ضمن أنظمة الزمن الحقيقي إن لم يتم إنشاؤهمسبقاً كمالابدّ من أن يتم هدمهبرمجياً بعد الانتهاء من العمل المطلوب منه.

- ▪ عدم استخدام الاسم البرمجي ذاته لمهمتين مختلفتين ضمن البرنامج نفسه.
- معالجة باقي تعليمات لغة C التي تُعد غير ضرورية في إنجاز ما هو مطلوب من مترجمنا. فعلياً قمنا بمناقشة التعليمات الأكثر استخداماً وتأثيراً في أنظمة الزمن الحقيقي بالشكل الذي يساعدنا على كتابة برامج نظامية ضمن لغة C/VxWorks وإنتاج مترجم يحقق الهدف المنشود من البحث على أكمل وجه أما تضمين كافة قواعد لغة C المختلفة فهذا أمر ثانوي تكميلي يمكن متابعته لاحقاً .
- هناك بعض التوابع الخاصة بعناصر الزمن الحقيقي التي لم يتم معالجتها ضمن مترجمنا لعدم تأثيرها الكبير في أداء هذه العناصر ولكنها قد تكون ضرورية لمتطلبات برمجية كثيرة، وقد تم إهمال هذه التوابع دون أن تؤثر على إكمال البحث. لقد كان هدفنا من هذا العمل صياغة أسلوب معالجة قابلة للاعتماد تفيد في عملية مقابلة تعليمات اللغتين بغض النظر عن الدراسة الشاملة لكافة التوابع المتاحة وفق لغة الدخل. إن معالجة التوابع المتبقية لاحقاً ضمن دراسات أخرى قد يكون مفيداً في إكمال شمولية تعليمات هذا المترجم. على سبيل المثال نذكر التوابع (الجدول7-1) التالية.

| التابع | الوظيفة البرمجية التي يؤديها |
|---|---|
| taskRegsGet( ) | تأخذ قيم مسجلات المهمة من جدول TCB. |
| taskRegsSet( ) | إسناد قيمة لمسجلات المهمة. |
| taskRegsShow( ) | إظهار محتوى مسجلات المهمة. |
| taskRestart( ) | إعادة تشغيل المهمة. |
| taskSwitchHookAdd( ) | إضافة برنامج ليتم استدعاؤه عند كل مرحلة تبديل للمهمة. |
| taskSwitchHookDelete( ) | حذف برنامج تبديل المهمة المضاف مسبقاً. |
| taskSwitchHookShow( ) | إظهار قائمة ببرامج تبديل المهمة. |
| taskTcb( ) | الحصول على كتلة التحكم بالمهمة المقابلة للمعرّف المحدد. |

**الجدول(7-1): توابع غير معالجة مقابلة لعنصر المهمة**

# الملحق /I/

# قواعد لغة C/VxWorks مصاغة وفق BNF

# قواعد لغة C/VxWorks مصاغة وفق BNF

يقدّم هذا الملحق عرضاً لأهم قواعد لغة C/VxWorks المصاغة وفق طريقة BNFوهي تُمثّل القواعد الأساسية التي تم الاعتماد عليها أثناء كتابة ملف التحليل التركيبي وفق تقنية Yacc لإنتاج مترجمنا المنشود.

**rogram:** header_statement declaration_list_variable Func_Prototype declaration_list_variable C_Vxworks_blocks;
**header_statement:** lab_declaration | const_declaration;
**lab_declaration:** lab_declaration library | library ;
**library :** "#" "include" sign_library identifier "." "h" sign_library;
**sign_library:** "<" | "Double_QU" | ">" |;
**const_declaration:** const_declaration const_declare |;
**const_declare:** "#" "define" identifier value | "const" | "const" identifier assignment_operator value;
**assignment_operator:** "=" | "*=" | "/=" | "%=" | "+=" | "-=" |;
**value:** Signed_Number | sign_library identifier sign_library | Letter_identifier;
**Letter_identifier:** Letter_identifier identifier_nondigit | identifier_nondigit;
**identifier :** identifier_nondigit | identifier identifier_nondigit | identifier Number |;
**identifier_nondigit:** "Schar";
**Digit :** "SNumber";
**Signed_Number :** sign Number | Number ;
**Number:** Number Digit | Digit ;
**sign:** "+" | "-";
**declaration_list_variable:** declaration_list_variable declaration_var | declaration_list_variable declaration_Types |;
**declaration_Types:** struct_type | array_type | enum_type ;
**struct_type:** "struct" identifier "{" declaration_list_variable "}" identifier ";";
**array_type:** simple_type identifier "[" size "]" | simple_type identifier "[" "]" | identifier "[" "]";
**size:** Number | identifier;
**simple_type :** "char" | bool_type | "short" | "int" | "long" | "signed" | "unsigned" | "float" | "double" | "void" | "struct" identifier ;
**bool_type:** "BOOL" identifier assignment_operator bool_value | "BOOL" identifier | identifier;
**bool_value:** "false" | "true";
**enum_type:** "enum" "{" list_identifier "}" ";";
**list_identifier:** list_identifier identifier comma | ;
**comma:** "," | ;
**declaration_var:** declaration_var declaration | declaration ;
**declaration:** permission simple_type_Declaration | RTS_variable ";" ;
**permission:** "LOCAL" | "Private" | "Public" | ;
**simple_type_Declaration:** simple_type identifier assignment_operator identifier | pointer_type | array_element_type;
**pointer_type:** simple_type "*" identifier | "*" identifier ;
**array_element_type:** identifier "[" Number "]";
**RTS_variable:** SEM_variable | Wd_Variable | MQ_Variable | vectorAddress | STATUS | Int_variable | Char_Variable | TCB_variable | bool_type | array_type | REG_variable | MSQ_INFO_variable;
**SEM_variable:** "SEM_ID" identifier | identifier ;
**Wd_Variable:** "WDOG_ID" identifier | identifier ;
**MQ_Variable:** "MSG_Q_ID" identifier | identifier ;
**vectorAddress:** " VOIDFUNCPTR" pointer_type identifier | identifier;

**STATUS:** "STATUS" |;
**Char_Variable:** "char" pointer_type identifier |"char" identifier | identifier ;
**TCB_variable:"** WIND_TCB" pointer_type identifier |" WIND_TCB " identifier | identifier ;
**REG_variable:** "REG_SET" pointer_type identifier |"REG_SET" identifier |identifier ;
**MSQ_INFO_variable:** "MSG_Q_INFO" pointer_type identifier | "MSG_Q_INFO" identifier | identifier;
**Func_Prototype:** Func_Prototype Func_declare |;
**Func_declare:** simple_type identifier "(" simple_type_Declaration ")" ";" ; // the identifier is Func_Name;
**C_Vxworks_blocks:** C_Vxworks_blocks Vxworks_blocks | Vxworks_blocks;
**Vxworks_blocks:** Func_Prototype "{" RTS_statement "}";
**RTS_statement:** RTS_statement task_declaration |RTS_statement Semaphore_declaration |RTS_statement interrupt_declaration |RTS_statement alarm_declaration |RTS_statement mailBox_declaration |;
**Func_RTS_Variable:** "VOIDFUNCPTR" identifier |"FUNCPTR" identifier | identifier ;
**task_var:** Int_variable ;
**POptions_var:** "VX_FP_TASK" | "VX_PRIVATE_ENV" | "VX_NO_STACK_FILL" |"VX_UNBREAKABLE";
**task_declaration:** Int_variable "=" "taskSpawn" "(" Char_Variable "," task_var "," task_var "," Func_RTS_Variable "," Int_variable ")"
|STATUS "taskInit" "(" TCB_variable "," Char_Variable "," task_var "," task_var "," Char_Variable "," Int_variable "," Func_RTS_Variable "," Int_variable ")"
|STATUS "taskDelete" "(" Int_variable ")"
|STATUS "taskSafe" "(" "void" ")"
|STATUS "taskUnsafe" "(" "void" ")"
|STATUS "taskPriorityGet" "(" Int_variable "," task_var ")"
|STATUS "taskPrioritySet" "(" Int_variable "," task_var ")"
|STATUS "taskSuspend" "(" Int_variable ")"
|STATUS "taskResume" "(" Int_variable ")"
|STATUS "taskDelay" "(" Int_variable ")"
|STATUS "taskIdVerify" "(" Int_variable ")"
|STATUS "taskActivate" "(" Int_variable ")"
|Int_variable "=" "taskIdSelf" "(" "void" ")"
|STATUS "taskRestart" "(" Int_variable ")"
|STATUS "taskLock" "(" "void" ")"
|STATUS "taskUnlock" "(" "void" ")"
|"void" "exit" "(" Int_variable ")"
|STATUS "taskDeleteForce" "(" Int_variable ")"
|STATUS "taskUnlock" "(" simple_type ")"
|TCB_variable "=" "taskTcb" "(" Int_variable ")"
|STATUS "taskOptionsSet" "(" Int_variable "," Int_variable "," task_var ")"
|STATUS "taskOptionsGet" "(" Int_variable "," POptions_var ")"
|STATUS "taskRegsGet" "(" Int_variable "," REG_variable ")"
|STATUS "taskRegsSet" "(" Int_variable "," REG_variable ")"
|Char_Variable "=" "taskName" "(" Int_variable ")"
|Int_variable "=" "taskNameToId" "(" Char_Variable ")"
|Int_variable "=" "taskIdDefault" "(" Int_variable ")"
|bool_type "=" "taskIsReady" "(" Int_variable ")"
|bool_type "=" "taskIsSuspended" "(" Int_variable ")"
|Int_variable "=" "taskIdListGet" "(" array_type "," Int_variable ")";
**Int_variable:** "int" pointer_type identifier |"int" identifier |"UINT" identifier |identifier |Number;
**Sem_State:** "SEM_FULL"|"SEM_EMPTY" ;
**Semaphore_declaration:** SEM_variable "=" "semBCreate" "(" Queue_State "," Sem_State ")"
|SEM_variable "=" "semCCreate" "(" Queue_State "," Int_variable ")"
|SEM_variable "=" "semMCreate" "(" QueueM_State ")"
|STATUS "semDelete" "(" SEM_variable ")"
|STATUS "semGive" "(" SEM_variable ")"
|STATUS "semFlush" "(" SEM_variable ")"
|STATUS "semTake" "(" SEM_variable "," Time_out ")"
|STATUS "semMGiveForce" "(" SEM_variable ")"

|SEM_variable "=" "semCSmCreate" "(" "SEM_Q_FIFO" "," Int_variable ")"
|"void" "semShowInit" "(" "void" ")"
|Int_variable "=" "semInfo" "(" SEM_variable "," array_type "," Int_variable ")"
|STATUS "semShow" "(" SEM_variable "," Int_variable ")";

**Queue_State:** "SEM_Q_PRIORITY"|"SEM_Q_FIFO" ;

**QueueM_State:** "SEM_Q_PRIORITY"|"SEM_Q_FIFO"|"SEM_DELETE_SAFE"|"SEM_INVERSION_SAFE" ;

**Time_out:** "WAIT_FOREVER"|"NO_WAIT"| Int_variable;

**alarm_declaration:** Wd_Variable "=" "wdCreate" "(" "void" ")" |STATUS "wdDelete" "(" Wd_Variable ")" |STATUS "wdStart" "(" Wd_Variable "," Int_variable "," Func_RTS_Variable "," Int_variable ")"
|STATUS "wdCancel" "(" Wd_Variable ")";

**MQ_Options :** "MSG_Q_FIFO"|"MSG_Q_PRIORITY";

**MQ_priority:** "MSG_PRI_NORMAL"|"MSG_PRI_URGENT";

**mailBox_declaration:** MQ_Variable "=" "msgQCreate" "(" Int_variable Int_variable "," MQ_Options ")"
|STATUS "msgQDelete" "(" MQ_Variable ")"
|STATUS "msgQSend" "(" MQ_Variable "," Char_Variable "," Int_variable "," Time_out "," MQ_priority")"
|Int_variable "=" "msgQReceive" "(" MQ_Variable "," Char_Variable "," Int_variable "," Time_out ")"
|MQ_Variable "=" "msgQSmCreate" "(" Int_variable Int_variable "," MQ_Options ")"
|"void" "msgQShowInit" "(" "void" ")"
|STATUS "msgQInfoGet" "(" MQ_Variable "," MSQ_INFO_variable ")"
|STATUS "msgQShow" "(" MQ_Variable "," Int_variable ")";

**interrupt_declaration:** STATUS "intConnect" "(" vectorAddress "," Func_RTS_Variable "," Int_variable ")"
|Int_variable "=" "intLock" "(" "void" ")"|"void" "intUnlock" "(" Int_variable ")"
|Int_variable "=" "intDisable" "(" Int_variable ")" |Int_variable "=" "intEnable" "(" Int_variable ")" ;

نلاحظ أن الطرف الأيسر من هذه القواعد يتضمن حصراً على رموز غير محددة (يتم تحليلها وفق قواعد أخرى) أما الطرف الأيمن من القواعد فقد يتضمن الرموز بنوعيها المحددة و غير المحددة علماً أن الرموز المحددة تكون عادة موجودة بين إشارتي الاقتباس وهي تُمثّل كلمات مفتاحية محجوزة للغة الدخل. على سبيل المثال يقابل الرمز program القاعدة الأولى في لغة الدخل C/VxWorks المُعبّرة عن الهيكلية العامة للبرنامج ويوضّح هذا الرمز أن برامج لغة الدخل تتكوّن من تعليمات استدعاء لملفات رأسية يليها التصريح عن المتحولات مع ترويسات التوابع المعرّفة ثم كتل برمجية أخرى.

# الملحق /II/

# المقابلة بين أهم توابع لغة C/VxWorks ولغة IF

# المقابلة بين أهم توابع لغة C/VxWorks ولغة IF

يتضمن هذا الملحق عرضاً لأهم توابع عناصر الزمن الحقيقي المتاحة وفق لغة C/VxWorks والشيفرة البرمجية المقابلة لها وفق لغة IF والناتجة عن الدراسة التحليلية التي تم إيضاحها وفق مخططات الأتومات المذكورة في الفصل الرابع. سنعرض كل عنصر مع أهم التوابع البرمجية الخاصة به وما يقابلها من تعليمات لغة IF. علماً أن جداول المقابلة المذكورة في هذا الملحق لاتتضمن كافة الحالات المعالجة وإنما أبرزها، كما يدل الرمز (...) على إمكانية وجود تعليمات أخرى لا تتعلق بالتابع المدروس.

## 1.II المهمة Task:

| Name's Function | The Format of Function | Equivalent statements under IF language |
|---|---|---|
| **taskInit** | STATUS taskInit( WIND_TCB * pTcb, char * name,int priority, int options, char * pStackBase, int stackSize, FUNCPTR entryPt, int arg1, int arg2, int arg3, int arg4, int arg5, int arg7, int arg8, int arg9, int arg10) | **System File:** signal taskInit(); |
| | | **state dormant** #start ;<br>input taskInit(); nextstate waiting;<br>endstat; |
| **taskActivate** | STATUS taskActivate(int tid) | signal taskActivate(); |
| | | **state waiting;**<br>input taskActivate(); nextstate ready;<br>endstat; |
| **taskSpawn** | int taskSpawn( char *name, int priority, int options, int stackSize, FUNCPTR entryPt, int arg1, int arg2, int arg3, int arg4, int arg5, int arg6, int arg7, int arg8, int arg9, int arg10) | signal taskSpawn(); |
| | | **state dormant** #start ;<br>input taskSpawn(); nextstate ready;<br>endstate; |
| **taskSuspend** | STATUS taskSuspend( int tid) | signal taskSuspend(); |
| | | input taskSuspend(); nextstate waiting; |
| **taskDelete** | STATUS taskDelete( int tid) | signal taskDelete(); |
| | | input taskDelete(); nextstate dormant; |
| **taskResume** | STATUS taskResume( int tid) | signal taskResume(); |
| | | input taskResume(); nextstate ready; |
| **taskDelay** | STATUS taskDelay( int ticks ) | **process task_name(1);**<br>var time clock; var task_delay integer; ...<br>state executing; ... nextstate waiting;<br>set time := 0 ;<br>nextstate pended; endstate;<br>**state pended;**<br>when time = task_delay; nextstate ready; ...<br>endstate; |

**الجدول(II−١): التوابع الأساسية للمهمة في VxWorks**

## 2.II المؤقت WatchDog (alarm):

| Name's Function | The Format of Function | Equivalent statements under IF language |
|---|---|---|
| **wdCreate** | WDOG_ID wdCreate(void) | **System File:** signal wdCreate(); signal create(); |
| | | **Process Alarm_File(1); ...**<br>**state dormant** #start ;<br>input wdCreate();<br>output create() to ({AlarmRoutine _File }0);<br>nextstate ready; endstate;<br>**Process AlarmRoutine _File(1);...**<br>**state dormant** #start ; input create(); nextstate ready; |
| **wdStart** | STATUS wdStart ( WDOG_ID wdId, int delay, FUNCPTR pRoutine, int parameter) | signal wdStart(integer); signal execute_routine(); |
| | | **Process Alarm_File(1);**<br>var alarm_delay integer; /* waiting time */ var alarm_time clock;<br>**state ready;**<br>input wdStart(alarm_delay); set alarm_time :=0; nextstate armed;<br>endstate;<br>**state armed;** ... when alarm_time = alarm_delay;<br>output execute_routine() to ({AlarmRoutine _File }0); nextstate ready;<br>endstate;<br>**Process AlarmRoutine _File(1);**<br>**state ready;** input execute_routine(); /*check_allowed_statements();*/<br>nextstate ready;<br>endstate; |
| **wdDelete** | STATUS wdDelete( WDOG_ID wdId) | signal wdDelete(); |
| | | input wdDelete(); nextstate dormant; |
| **wdCancel** | STATUS wdCancel ( WDOG_ID wdId) | signal wdCancel(); |
| | | **state armed;** input wdCancel(); reset alarm_time; nextstate ready; ... |

**الجدول(II−٢): التوابع الأساسية للمؤقت في VxWorks**

## 3.II السيمافور الثنائي Binary Semaphore:

| Name's Function | The Format of Function | Equivalent statements under IF language |
|---|---|---|
| **semTake** | STATUS semTake (SEM_ID semId,int timeout) | **signal semBTake(pid,integer);** /* sem_delay variable may be:<br>1.WAIT_FOREVER(-1) 2. NO_WAIT(-2) 3. TIME (any integer number) */<br>signal Ack_SemTake(pid); signal Ack_SemError(pid); |

| Name's Function | The Format of Function | Equivalent statements under IF language |
|---|---|---|
| | | **state empty;** input semBTake(task_ID,sem_delay);<br>nextstate decision_2;<br>**state decision_2** #unstable ;<br>provided (sem_delay = -2); /* NO_WAIT */<br>output Ack_SemError(task_ID) to ({ID_sentTask}0); nextstate empty;<br>provided (sem_delay >= 0); /* time value */ set time:=0;<br>nextstate waiting_withTime;<br>provided (sem_delay = -1); /* WAIT_FOREVERE */<br>nextstate waiting_full;<br>endstate;<br>**state waiting_withTime;**<br>when time = sem_delay;<br>output Ack_SemError(self) to ({ID_sentTask}0);<br>nextstate empty;… |
| **semGive** | STATUS semGive (SEM_ID semId ) | signal semGive(pid); |
| | | **state full;** input semGive(task_ID); nextstate full; endstate;<br>**state waiting_full;** input semGive(task_ID);<br>output Ack_SemTake(self) to ({ID_sentTask}0);<br>nextstate empty; endstate;<br>**state empty;** input semGive(task_ID); nextstate full; endstate;<br>**state waiting_withTime;**<br>input semGive(task_ID); output Ack_SemTake(self) to ({ID_sentTask}0);<br>nextstate empty;<br>endstate; |
| **semDelete** | STATUS semDelete (SEM_ID semId) | signal semDelete(pid); |
| | | input semDelete(task_ID); nextstate dormant; |
| **semBCreate** | SEM_ID semBCreate ( int options, SEM_B_STATE initialState ) | type Initial_state = enum SEM_EMPTY,SEM_FULL endenum;<br>signal semBcreate(pid,Initial_state); |
| | | var bin_Semaphore Initial_state; var sem_delay integer;<br>var time clock; var task_ID pid;<br>**state dormant** #start ;<br>input semBcreate(task_ID,bin_Semaphore);<br>nextstate decision_1; endstate;<br>**state decision_1** #unstable ;<br>provided (bin_Semaphore = SEM_FULL); nextstate full;<br>provided (bin_Semaphore = SEM_EMPTY);<br>nextstate empty; endstate;<br>**state full;** input semBTake(task_ID,sem_delay);<br>output Ack_SemTake(self) to ({ID_sentTask}0);<br>nextstate empty; endstate; |
| **semCCreate** | SEM_ID semCCreate (int options, int initialCount ) | signal semCcreate(pid); |
| | | const max_entities = 50;<br>var time clock; var NumRequest integer; var counter integer; var task_Id pid;<br>**state dormant** #start ; input semCcreate(task_Id);<br>task counter := 0;<br>task NumRequest := 0; |
| **semMCreate** | SEM_ID semMCreate (int options) | signal semMcreate(pid); |
| | | **state dormant** #start ;<br>input semMcreate(task_Id);<br>nextstate full; endstate; |

**الجدول(II-٣): التوابع الأساسية للسيمافور الثنائي في VxWorks**

## 4.II صندوق البريد (رتل الرسائل) MessageQueue:

أهم ما يميز عمل علبة البريد ضمن نظام VxWorks هو التوابع البرمجية التالية (الجدول II-4) التي تم مقابلتها بتعليمات لغة IF، حيث ناقشنا كافة القيم الزمنية المسموحة للاستجابة المطلوبة من العلبة وذلك من خلال التصريح عن متحولات زمنية (msg_delay, msg_ReceivedDelay, msg_time) ذات دلالات مختلفة تم إيضاحها سابقاً في الفصل الرابع.

| Name's Function | The Format of Function | Equivalent statements under IF language |
|---|---|---|
| **msgQCreate** | MSG_Q_ID msgQCreate(int maxMsgs, int maxMsgLength, int options ) | signal msgQCreate(integer); |
| | | **state dormant** #start ;<br>input msgQCreate(max_msgs);<br>task msg_number := 0;<br>nextstate empty;<br>endstate; |
| **msgQDelete** | STATUS msgQDelete( MSG_Q_ID msgQId) | signal msgQDelete(); |
| | | input msgQDelete(); nextstate dormant; |
| **msgQSend** | | signal msgQSend(integer); |

| Name's Function | The Format of Function | Equivalent statements under IF language |
|---|---|---|
| | STATUS msgQSend(MSG_Q_ID msgQId, char * buffer, UINT nBytes, int timeout, int priority ) | var msg_number integer; var max_msgs integer<br>var msg_delay integer; /* time senddelay */;<br>var msg_ReceivedDelay integer; var msg_time clock;<br>**state empty;**<br>input msgQSend(msg_delay);<br>task msg_number := msg_number + 1;<br>nextstate Not_empty; …<br>**state Not_empty;**<br>input msgQSend(msg_delay); task msg_number := msg_number + 1;<br>nextstate decision_1; endstate;<br>**state decision_1** #unstable ;<br>provided (msg_number < max_msgs);<br>nextstate Not_empty;<br>provided (msg_number = max_msgs); output ack() to ({ID_sentTask}0);<br>nextstate full; endstate;<br>**state full;**<br>input msgQSend(msg_delay);<br>task msg_number := msg_number + 1; nextstate decision_3;<br>endstate;<br>**state decision_3** #unstable ;<br>provided (msg_delay = -1); /* WAIT_FOREVER */<br>nextstate wait_acceptance;<br>provided msg_delay = -2; /* No_Wait */<br>task msg_number := msg_number - 1;<br>output error_msg() to ({ID_sentTask}0); nextstate full;<br>provided (msg_delay >= 0); /* Any_value */ reset msg_time;<br>nextstate wait_delaytime; endstate;<br>**state wait_Full;**<br>input msgQSend(msg_delay); /* with any type of time */<br>output msg_FirstWaiting() to ({ID_sentTask}0);<br>output ack() to ({ID_sentTask}0); nextstate empty;<br>endstate;<br>**state wait_FullWTime;**<br>input msgQSend(msg_delay); /* with any type of time */<br>output msg_FirstWaiting() to ({ID_sentTask}0); output ack() to ({ID_sentTask}0); nextstate empty;<br>when msg_time = msg_ReceivedDelay;<br>output error_msg() to ({ID_sentTask}0); nextstate empty;<br>endstate; |
| **msgQReceive** | | signal msgQReceive(integer);<br>/*msg_delay variable: 1. WAIT_FOREVER(-1) 2. NO_WAIT(-2) 3. TIME (any integer number)*/ |

| Name's Function | The Format of Function | Equivalent statements under IF language |
|---|---|---|
| **msgQReceive** | int msgQReceive( MSG_Q_ID msgQId, char * buffer, UINT maxNBytes, int timeout) | **state empty;**<br>input msgQReceive(msg_ReceivedDelay); nextstate decision_0; endstate;<br>**state decision_0;** provided (msg_ReceivedDelay >=0); /* time value */<br>set msg_time := 0; nextstate wait_FullWTime;<br>provided (msg_ReceivedDelay = -1); /* WAIT_FOREVERE */<br>nextstate wait_Full;<br>provided (msg_ReceivedDelay = -2); /* NO_WAIT */<br>output error_msg() to ({ID_sentTask}0); nextstate empty; endstate;<br>**state Not_empty;**<br>input msgQReceive(msg_ReceivedDelay); /* time of any type */<br>task msg_number := msg_number - 1; nextstate decision_2; ... endstate;<br>**state decision_2** #unstable ; provided msg_number >1; nextstate Not_empty;<br>provided msg_number = 1; nextstate empty;<br>provided msg_number = max_msgs; nextstate full; endstate;<br>**state wait_acceptance;**<br>input msgQReceive(msg_ReceivedDelay);<br>task msg_number := msg_number - 1;<br>output msg_FirstWaiting() to ({ID_sentTask}0); nextstate full; endstate;<br>**state wait_delaytime;**<br>when msg_time = msg_delay; task msg_number := msg_number - 1;<br>output error_msg() to ({ID_sentTask}0); nextstate full;<br>input msgQReceive(msg_ReceivedDelay);<br>task msg_number:=msg_number - 1;<br>nextstate decision_4; endstate;<br>**state decision_4** #unstable ;<br>when msg_time < msg_delay ;<br>output msg_FirstWaiting() to ({ID_sentTask}0); nextstate full;endstate; |

**الجدول(II–4): التوابع الأساسية لرتل الرسائل في VxWorks**

# 5.II برنامج خدمة المقاطعة ISR (Interrupt Service Routine):

| Name's Function | The Format of Function | Equivalent statements under IF language |
|---|---|---|
| **intConnect** | STATUS intConnect (VOIDFUNCPTR * vector, VOIDFUNCPTR routine, int parameter) | signal intConnect(); signal interrupt();<br>signalroute env_ISR(1) from env to ISR with interrupt; |
| | | **process ISR(1);**<br>var ISR_lockKey integer;<br>**state dormant** #start ;<br>input intConnect(); nextstate ready; endstate;<br>**state ready;** input interrupt(); nextstate executing; endstate;<br>**state masked;** input interrupt(); nextstate masked; endstate;<br>**state executing;** /* check_allowed_statements */<br>nextstate ready; endstate; |
| **intEnable** | int intEnable ( int level ) | signal intEnable(); |
| | | input intEnable(); nextstate ready; |
| **intDisable** | int intDisable ( int level ) | signal intDisable(); |
| | | input intDisable(); nextstate masked; |
| **intUnlock** | void intUnlock ( int lockKey ) | signal intUNLock(integer); |
| | | input intUNLock(ISR_lockKey); nextstate ready; |

**الجدول(II–5): التوابع الأساسية لبرنامج خدمة المقاطعة في VxWorks**

# الملحق /III/

# واجهات التطبيق

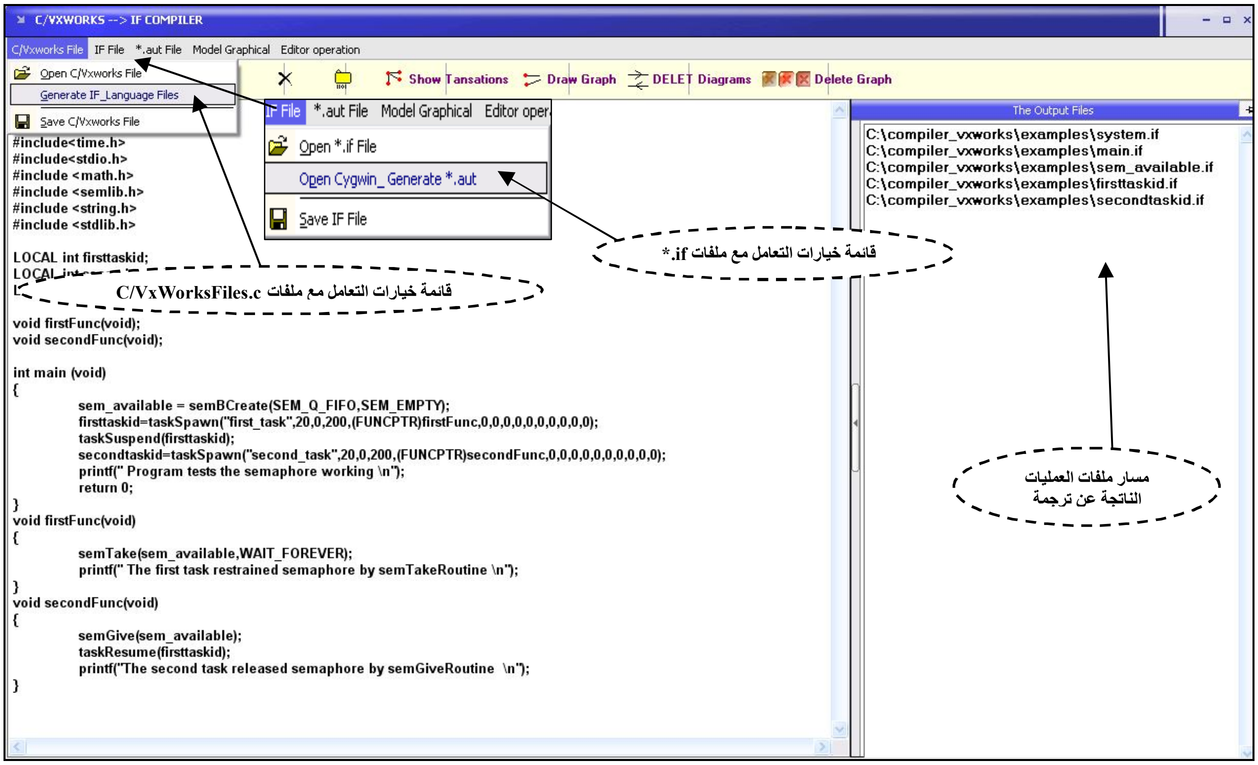
# واجهات التطبيق

يتضمن هذا الملحق شرحاً توضيحياً لواجهات المترجم وواجهات التطبيق الخاص بعمليات المعالجة التي تتبع عملية الترجمة، ليكون هذا الشرح بمثابة توثيق ودليل للمستخدم (المبرمج) يوضح له كيفية الاستخدام الفعّال الذي يساعده على استثماره عملياً بشكل صحيح والاستفادة منه في التحقق من صحة برمجياته، وبالتالي فإن بحثنا هذا بأفكاره المتعددة والمتشعبة لا يقتصر على كونه دراسة نظرية مُعدّة لنيل درجة الماجستير فحسب وإنما هو نتاج برمجي قابل للاستخدام والتطبيق العملي الفعّال الذي يوفر على مبرمج تطبيقات الزمن الحقيقي الكثير من الاختبارات المطلوبة للتحقق من صحة الشيفرة البرمجية ويساعده على تنفس الصعداء والثقة بمنتجه البرمجي قبل طرحه في سوق العمل.

## 1.III واجهة المترجم:

تُتيح الواجهة الرئيسية للمترجم (الشكل III-1) إمكانية التعامل مع ملفات ذات امتدادات مختلفة (مكتوبة بلغة C/VxWorks أو بلغة IF إضافة إلى ملفات المحاكاة الشاملة aut.*).

يُمكننا من خلال قائمة (C/VxWorksFile) فتح برامج C/VxWorks المطلوب ترجمتها وفق الخيار (Generate IF_Language Files) الذي يتم من خلاله إيجاد ملفات العمليات المستقلة المكوّنة للنظام المقابل لبرنامج C/VxWorks المُترجم، ومن ثم يتم تشكيل ملف النظام النهائي ذي الاسم (system.if) الناتج عن تجميع كافة الملفات السابقة والمتضمنة لعمليات النظام مع الأخذ بعين الاعتبار قواعد البرمجة وفق لغة IF (مثلاً يُمنع التصريح عن المتحولات المحلية الخاصة بكل عملية ضمن ملف النظام النهائي أكثر من مرة واحدة في حال تشاركها بين العمليات)، كما يمكننا فتح ملف النظام قيد المعالجة وفق الخيار (Open *.if File) التابع لقائمة (IF File) لمعالجته بإحدى أدوات لغة IF والحصول على ملف المحاكاة الشامل وذلك باختيار (Open Cygwin_Generate *.aut) وتطبيق عملية الـ making على ملف النظام الناتج.



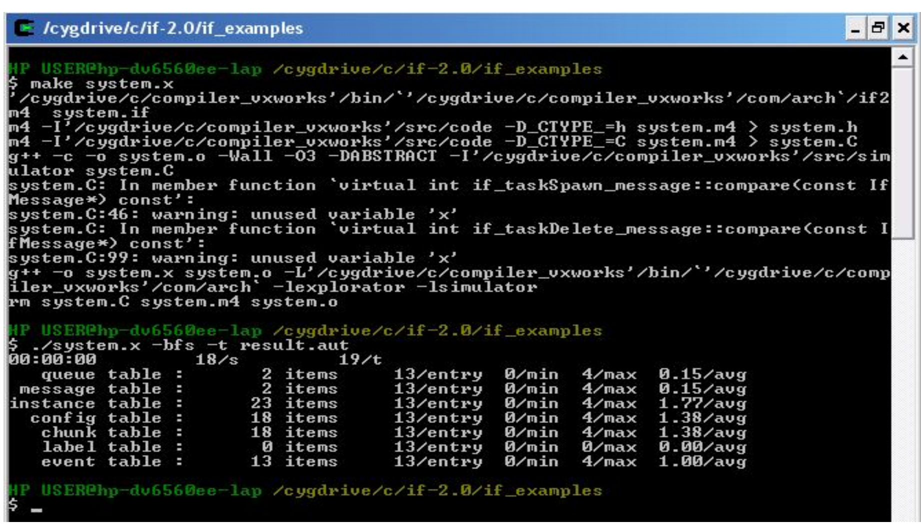

**الشكل (III-1): واجهة المترجم (C/VxWorks to IF)**

يتم إجراء عملية الـ making على ملف النظام المعالج بتطبيق التعليمة التالية:

> make system.x

نحصل بالنتيجة على ملف المحاكاة الشاملة المتضمن للحالات والمناقلات المقابلة لكافة وضعيات التنفيذ الممكنة، يتم توليد هذا الملف (الشكل III-2) باسم اختياري مثلاً (ExhaustiveSimulation.aut) يُحدّد من قبل المستخدم من خلال التعليمة التالية:

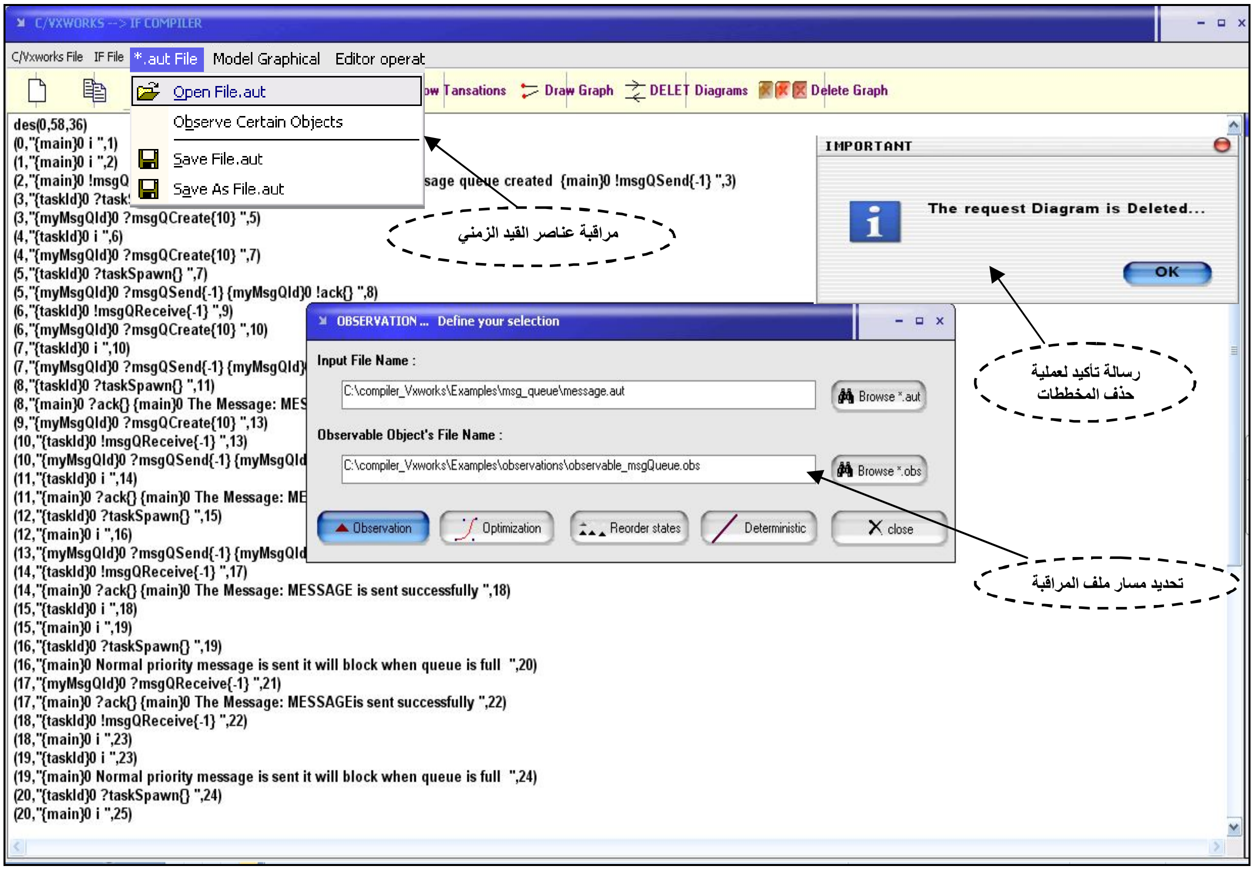
>./system.x –bfs –t **ExhaustiveSimulation**.aut




**الشكل(III-2): توليد ملف المحاكاة الشاملة وفق Cygwin**

## 2.III الواجهات التطبيقية:

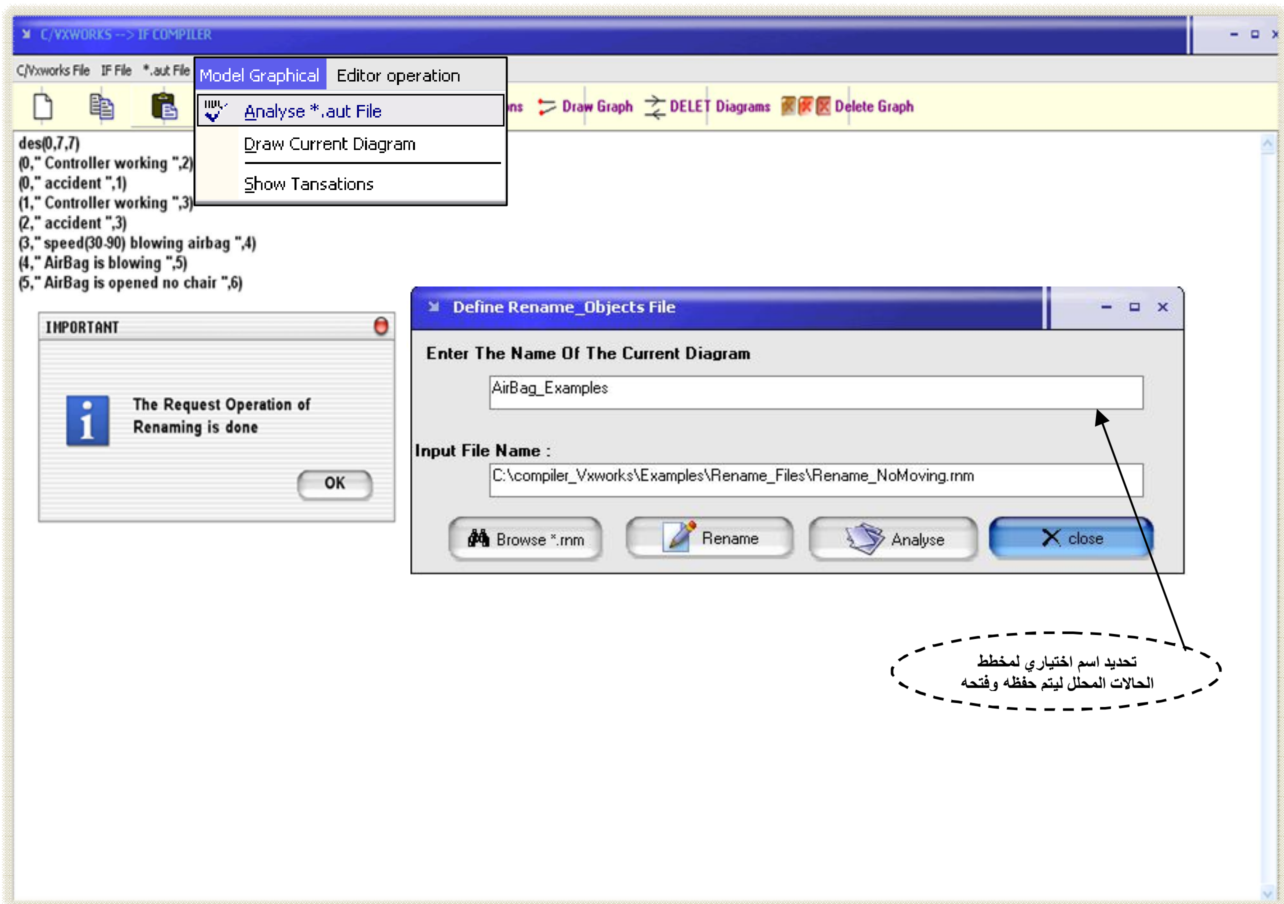
بعد قيام مترجمنا بعملية الترجمة المطلوبة ومعالجة برنامج الخرج وفق أو ات لغة IF، نحصل على مخطط الحالات الشامل الذي سيخضع لعدة معالجات ليتم إنتاج المخطط النهائي المتضمن للمسار المقابل للقيد الزمني المدروس (كما شرحناسابقاً). لتحقيق ذلك يتم أولاً فتح ملف المحاكاة الشامل (الشكل III-3) من خلال الخيار (Open*.aut File) ومن ثم تحديد مسار ملف المراقبة الخارجيالمتضمن عناصر المُراقبة (عملياتوا إشارات وعبارات خاصة) وذلك باختيار (Observe certain Objects) من القائمة نفسها حيث تظهر لنا نافذة يتم وفقها إتاحة كافة خيارات المعالجة المطلوبة، يمكننا من خلال هذه النافذة تحديد ملف الدخل المطلوب مراقبة مناقلاته(في حال كان حجمه كبيراً جداً) وملف المراقبة الخارجي (Observable_Objects.obs) ومن ثم تطبيق عملية المراقبة وفق الخيار (Observation) أما عملية تحسين المخطط الناتج (إلغاء المناقلات المتكررة) فتتم باختيار (Optimization) ومن ثم يمكننا إعادة ترقيم الحالات وفق الخيار (Reorder States)وقد تُتبع هذه العملية باختيار (Deterministic) لإزالة غموض المخطط غير المحدد فينتج لدينا ملف المحاكاة النهائي الذي يتم ترقيم حالاته ضمنياً وبشكل تلقائي.




الشكل (III-3): معالجة ملف المحاكاة الشامل (مراقبة، تحسين، تحديد المناقلات)

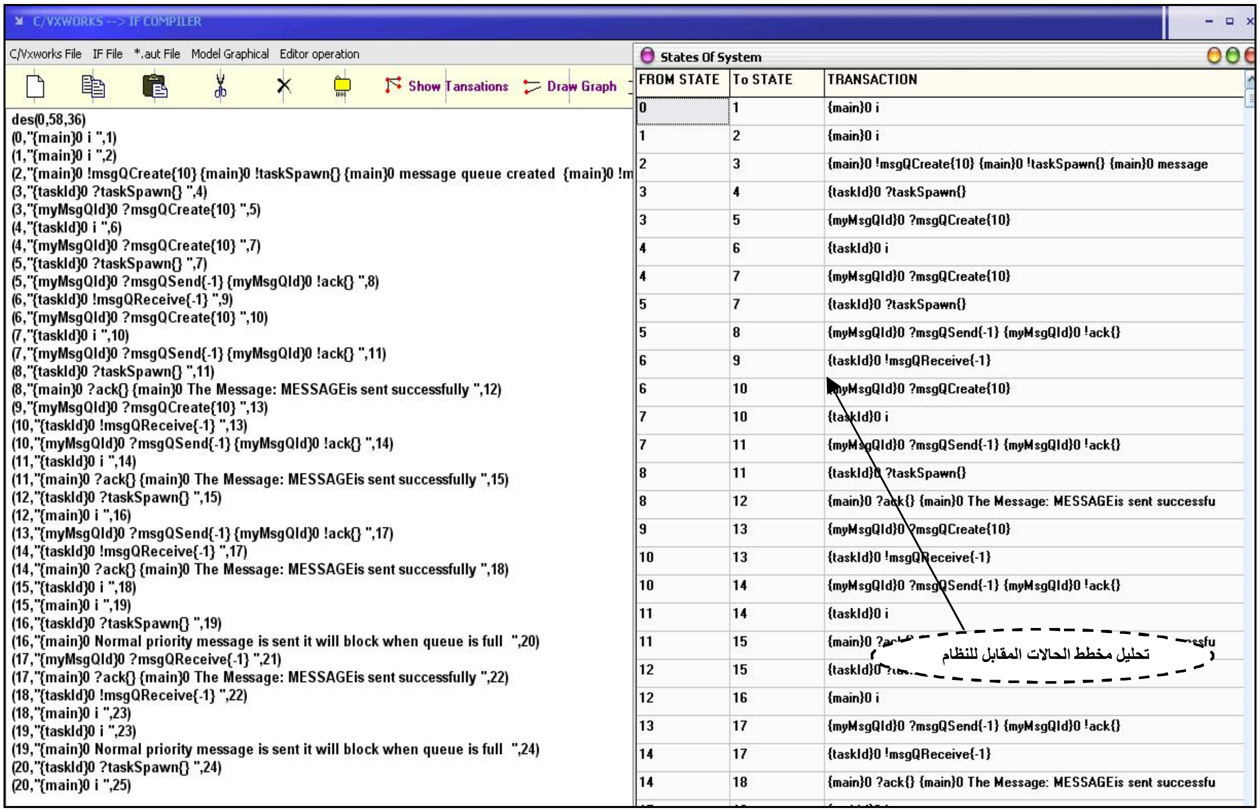
بعد معالجة ملف المحاكاة الشامل وفق الخطوات السابقة وإنتاج الملف النهائي يمكننا رسم المخطط البياني المقابل لهذا الملف وذلك بتحليله أولاً وفق الخيار (Analyse *.aut File) المُتاح من القائمة (Model Graphical) فتظهر لنا نافذة حوار خاصة تطلب من المستخدم تحديد اسم معين للمخطط المعالج (تُحفظ كافة المخططات المعالجة ضمن قاعدة بيانات خاصة يُتاح إعادة فتحها والتعامل معها لاحقاً).

يُتاح للمستخدم وفق هذه النافذة (الشكل III-4) القيام بعملية إعادة تسمية المناقلات من خلال تحديد ملف خارجي (Browse*.rnm) يتضمن المناقلات المطلوب إعادة تسميتها مع خيارات التسمية البديلة وتطبيق عملية إعادة التسمية هذه وفق الخيار (Rename) ومن ثم يتم تحليل الملف وفق الخيار (Analyse).



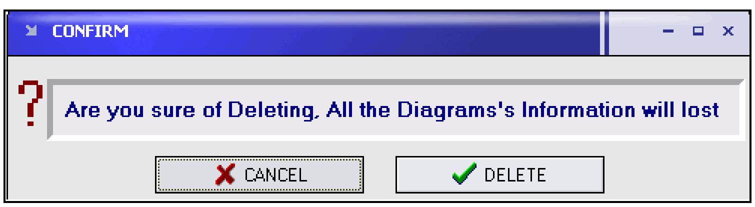

**الشكل (III-4): تطبيق إعادة التسمية وتحليل الملف قبل رسم مخططه**

يمكن عرض نتائج التحليل باختيار (Show Tansitions) من القائمة نفسها فيظهر جدول عام (الشكل III-5) بكافة مناقلات المخطط مع المواصفات المقابلة لكل منها (عقدة المنطلق From-State، التوصيف Transition، عقدة المستقر To-state).



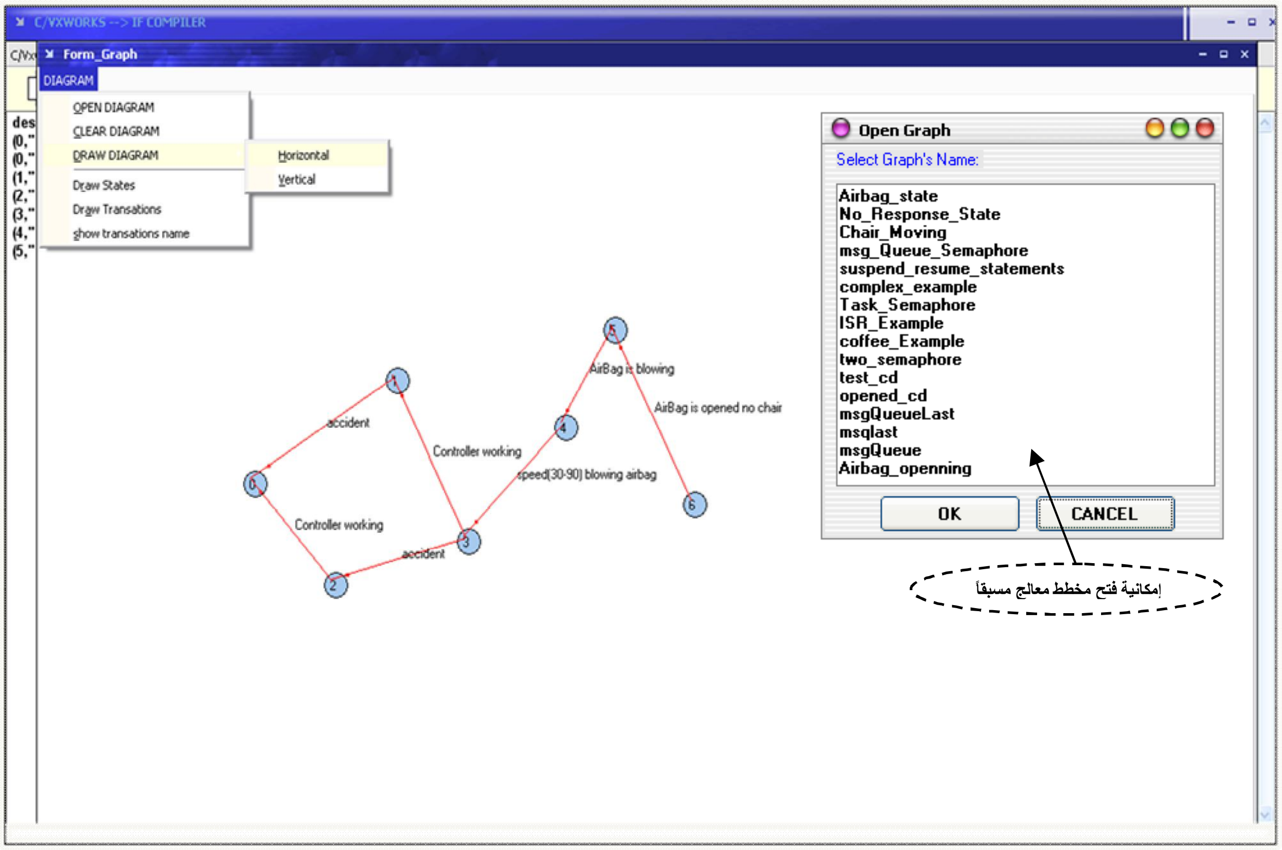

**الشكل (III-5): تحليل مخطط المحاكاة الشاملة**

كما يمكننا وفق هذا التطبيق القيام بحذف المخططات المعالجة مسبقاً (الشكل III-6).




**الشكل (III-6): حذف كافة معلومات المخططات المعالجة**

أخيراً أتحنا عملية الرسم البياني للمخطط النهائي المتضمن للقيود الزمنية المطلوبة باختيار (Draw Current Diagram) من القائمة السابقة (Model Graphical) حيث تظهر واجهة (الشكل III-7) يتم وفقها اختيار القائمة (Diagram) ومن ثم (OPEN DIAGRAM) ليتم عرض قائمة بالمخططات المعالجةمسبقاً وتحديد المخطط المطلوب رسمه مع تحديد استراتيجية الرسم المطلوبة (وفق المحور الأفقي Horizontal أو الشاقولي Vertical).




**الشكل (7-III): الرسم البياني للمخطط**

# الملحق /IV/

# الشيفرة البرمجية المقابلة لخوارزميات المعالجة

# الشيفرة البرمجية المقابلة لخوارزميات المعالجة

نستعرض في هذا الملحق جزءاً بسيطاً من الشيفرة البرمجية المقابلة لخوارزميات المعالجة التي تم ذكرها في الفصل السادس بعد أن تمت برمجتها وفق لغة Delphi.

### 1. process_deleted_line:

يقوم هذاالإجراء بقراءة ملف المحاكاة الشامل وتحديد المناقلات غير المطلوبة لتشكيل المجموعات المحذوفة التي سيتم الاعتماد عليها ضمن الخطوة الأولى من عملية المراقبةتلي تُطبّق على المخطط.

```
procedure process_deleted_line(line_info:string);
      var first_node,second_node:string;   new_count,j,len,row,column:integer;  added_node:boolean;
Begin  { This procedure defines the deleted groups that are formed as the first step at the observation operation  }
        added_node:=false;  len:=length(line_info);   j:=2;
        while (j<=len-1) do
        Begin
               row:=0; column:=0; first_node:=''; second_node:='';
               while (line_info[j]in ['0'..'9'])and (j<=len-1)  do
               Begin {define the first node from the processing transition}
                         first_node:=first_node+ line_info[j];   j:=j+1;
                end;
                j:=j+2;
               while (line_info[j]<>'''')and (j<=len-1)  do
                            j:=j+1;
                j:=j+2; //abort comma
               while (line_info[j]in ['0'..'9'])and (j<=len-1)  do
               Begin {define the second node from the processing transition}
                       second_node:=second_node+ line_info[j];   j:=j+1;
               end;
               if (second_node<>'') and  (first_node<>'') then
                          if   not added_node and node_founded_certainGroup(strtoint(first_node),row,column) then
                          begin      // add the second node to the correct group
                                 new_count:=increase_num_elements_Row(row);
                                 setlength(Deleted_nodes_optimizedFile[row],new_count);
                                 Deleted_nodes_optimizedFile[row,new_count-1]:=strtoint(second_node);  added_node:=true;
                          end;
               if not added_node and node_founded_certainGroup(strtoint(second_node),row,column) then
               begin          // add the first node to the correct group
                       new_count:=increase_num_elements_Row(row);
                       setlength(Deleted_nodes_optimizedFile[row],new_count);
                       Deleted_nodes_optimizedFile[row,new_count-1]:=strtoint(first_node);
                       added_node:=true;
               end;
               if  not added_node then  // not found the two nodes into any group , it will forms new group
               begin
```

```
                num_row_Deleted_nodes_optimizedFile:= num_row_Deleted_nodes_optimizedFile+1;
                setlength(Deleted_nodes_optimizedFile,num_row_Deleted_nodes_optimizedFile);
                setlength(Deleted_nodes_optimizedFile[num_row_Deleted_nodes_optimizedFile-1],2);
                Deleted_nodes_optimizedFile[num_row_Deleted_nodes_optimizedFile-1,0]:=strtoint(first_node);
                Deleted_nodes_optimizedFile[num_row_Deleted_nodes_optimizedFile-1,1]:=strtoint(second_node);
                increase_num_elements_Row(num_row_Deleted_nodes_optimizedFile-1);
                increase_num_elements_Row(num_row_Deleted_nodes_optimizedFile-1);
          end;
          j:=j+1;
    end;// end while
end;
```

## 2. replace_equivalent_nodes:

يتم وفق هذا التابع نقل المناقلات غير المحذوفة إلى الملف بعد معالجة عقدتي طرفيها وتعديلهما بالشكل المناسب، بحيثتُستبدل العقدة التي تنتمي إلى إحدى المجموعات المحذوفة باسم المجموعة المكافئة. يُقابل هذا التابع الخطوة الثانية في خوارزمية المراقبة.

```
function replace_equivalent_nodes(line:string):string;
    var i,j,len,index_request,row_founded,col_founded:integer; buffer:string;
Begin { replace each node of the transition into file with the equivalent group or keeping it}
      len:=length(line);   j:=2;  result:='(';
     while (j<=len-1) do
     Begin
           row_founded:=0; col_founded :=0;  buffer:='';
           while (line[j]in ['0'..'9']) and  (j<=len-1)  do
           Begin
                 buffer:=buffer+ line[j];   j:=j+1;
           end;
           if buffer<>'' then //replace first_node(_from_)
          begin
          if node_founded_certainGroup(strtoint(buffer),row_founded,col_founded) then
                buffer:=inttostr(100+row_founded);
          end;
               result:=result+buffer+line[j]; //add coma
               j:=j+2;
               result:=result+ line[j-1]; //add the symbol "
               buffer:='';
               while (line[j]<>'"')and (j<=len-1)  do // complete the information of line to send it to the called
               begin
                     result:=result+line[j];    j:=j+1;
               end;
               result:=result+ line[j];  j:=j+2; //abort comma
               result:=result+ line[j-1];
               while (line[j]in ['0'..'9'])and (j<=len-1)  do
               begin
                     buffer:=buffer+ line[j];   j:=j+1;
               end;
               if buffer<>'' then //replace first_node(_from_)
               begin
                     if node_founded_certainGroup(strtoint(buffer),row_founded,col_founded) then
                           buffer:=inttostr(100+row_founded);
               end;
               result:=result+buffer+')' ; //+ replaced number
```

```
                j:=j+1;
    end;   end;
```

### 3. btn_determineClick:

يتم وفق هذا الإجراء القيام بعملية إزالة غموض الأتومات غير المحدد وتحويله إلى أتومات محدد.

```
procedure TForm_Observation.btn_determineClick(Sender: TObject);
    var deterministic_file_name,s,node,first,second:string; deterministic_array:array[1..50,1..3]of string;
    SL:TStringList;  i,j,len,count_lines,index:integer;
Begin                 { this code converts the undeterministic automata to deterministic }
   deterministic_file_name:=input_file_observable +'_OBS'+inttostr(num_execution_step)+'_OPT'+
        inttostr(num_execution_step)+'_Final'+'.aut';
   if  not fileexists(uppercase(deterministic_file_name)) then   // show warning message
   begin
        form_main.Message_note.Text:='You must generate the request file before';
        form_main.Message_note.ShowModal;
   end
   else
   begin
        SL := TStringList.Create;   SL.LoadFromFile(deterministic_file_name);
        deterministic_file_name:=input_file_observable +'_OBS'+inttostr(num_execution_step)+'_OPT'+
             inttostr(num_execution_step)+ '_Final_D'+'.aut';
        if  fileexists(uppercase(deterministic_file_name)) then   // check if the file was created before, delete it
        begin
             DeleteFile(deterministic_file_name);    create_File(deterministic_file_name);
        end
        else
        create_File(deterministic_file_name);   count_lines:=  sl.Count-1;
        for i:=1 to count_lines do  // for every line into file analyse it
        begin
             s:=sl.Strings[i];  len:=length(s);  j:=2;  node:='';
             while (j<=len-1) do
             Begin
                    while (s[j]in ['0'..'9'])and (j<=len-1)  do
                    Begin
                           node:=node+ s[j];     j:=j+1;
                    end;
                    deterministic_array[i,1]:= node;    j:=j+2;   node:='';
                    while (s[j]<>'"')and (j<=len-1)  do
                    Begin
                          node:=node+ s[j];    j:=j+1;
                    End;
                    deterministic_array[i,2]:= node;   node:='';   j:=j+2; //abort comma
                    while (s[j]in ['0'..'9'])and (j<=len-1)  do
                    Begin
                          node:=node+ s[j];      j:=j+1;
                    end;
                    deterministic_array[i,3]:= node;     j:=j+1;
           end;// end while
       end;
    FreeAndNil(SL);
    for i:=1 to count_lines-1 do  // define the transitions that make the automata is undeterministic and process them
      for j:=i+1 to count_lines do
      Begin
       if (deterministic_array[i,1]=deterministic_array[j,1])and (deterministic_array[i,2]=deterministic_array[j,2]) then
```

```
        Begin
            first:=deterministic_array[i,3] ;     second:= deterministic_array[j,3];
            deterministic_array[i,3]:= deterministic_array[j,3];  // merge the two start_node
            deterministic_array[i,1]:='';
            for index:=1 to count_lines do //change all the tansitions that its start_node equals the deleted state
              if deterministic_array[index,1]=first then
                  deterministic_array[index,1]:=second;
        End;
      End;
      for i:=1 to count_lines do
         if  deterministic_array[i,1]<>'' then
           write_optimize_File(deterministic_file_name,'('+ deterministic_array[i,1]+','' +deterministic_array[i,2]+'','
             +deterministic_array[i,3]+')');
         current_filename:= deterministic_file_name;  Form_Main.SpeedButton3Click(Sender);
         deterministic_file_name:=define_name_file_of_path(deterministic_file_name);
         form_main.Message_note.Text:='The Final file '+deterministic_file_name;
         form_main.Message_note.ShowModal;
  end;
end;
```

### 4. Reorder_statesClick:

تُمثّل الشيفرة البرمجية التالية الجزء الأساسي من تعليمات التابع الذي يتم وفقه إعادة ترقيم حالات المناقلات الناتجة في ملف المحاكاة النهائي الذي تم مراقبة مناقلاتهوتحسينه أوا ٍزالة غموضه.

```
… for i:=0 to sl.Count-1 do  { for every transition into the file reorder its nodes }
    begin //replace the number into all the lines
        if  counter_states=2*i  then
             Form_progress.Gauge1.Progress :=i;
         s:=sl.Strings[i];   len:=length(s);    j:=2;  found_number:=false;
         while (j<=len-1) do
         Begin
             buffer:='';
             while (s[j]in ['0'..'9'])and (j<=len-1)  do
             Begin
                  buffer:=buffer+ s[j];   j:=j+1;
             end;
             if buffer<>'' then
             Begin
                  for index_row:=1 to 100  do
                  Begin
                       if ordered_numbers[index_row,1]=buffer then
                       Begin
                            first_node:=ordered_numbers[index_row,2]; found_number:=true ; break;
                       end;
                  end;
                  if not found_number then
                  Begin
                       Num_Nodes_arrangement:=Num_Nodes_arrangement+1;
                       first_node:=inttostr(Num_Nodes_arrangement);  activate_index:=activate_index+1;
                       ordered_numbers[activate_index,1]:=buffer;
                       ordered_numbers[activate_index,2]:=inttostr(Num_Nodes_arrangement);
                  End;
             End;
             if buffer<>'' then
```

```
            begin
                 buffer:='';    found_number:=false;
                 while (s[j]<>'"')and (j<=len-1)  do    //increse index until equal the symbol "
                    j:=j+1;  j:=j+1; //increse index above equal symbol "
            end;
            while (s[j]<>'"')and (j<=len-1)  do
                j:=j+1;  j:=j+2; //abort comma and "
            while (s[j]in ['0'..'9'])and (j<=len-1)  do
            Begin
               buffer:=buffer+ s[j];       j:=j+1;
             end;
             if buffer<>'' then
             Begin
                 for index_row:=1 to 100  do
                 begin
                       if ordered_numbers[index_row,1]=buffer then
                       Begin
                         Second_node:=ordered_numbers[index_row,2];    found_number:=true; break;
                        end;
                 end;
                 if not found_number then    {Increase the total number of states}
                 Begin
                        Num_Nodes_arrangement:=Num_Nodes_arrangement+1;
                        second_node:=inttostr(Num_Nodes_arrangement); activate_index:=activate_index+1;
                         ordered_numbers[activate_index,1]:=buffer;
                         ordered_numbers[activate_index,2]:=inttostr(Num_Nodes_arrangement);
                 End;
            End;
    { replace the node with the correct arrangement}
            out_str:=replace_nodes(first_node,second_node,s);  SL[i]:=out_str; j:=j+1;
        End;// end while
   End; //end for
```

# المراجع:

[1] **Mostefa Belarbi, Jean-Philippe Babau, Jean-Jacques Schwarz**: *Temporal Verification of Real-Time Multitasking Application Properties Based on Communicating Timed Automata, Distributed Simulation and Real-Time Application*. Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real-Time Applications table of contents,2004 . ISBN ~ ISSN:1550-6525 , 0-7695-2232-7. PP 188-195.

[2]. **Joseph, Mathai.** *Real-time Systems Specification, Verification and Analysis.* s.l. : Tata Research Development & Design Centre, June 2001. p. 290 P. Original edition published in 1996 by Prentice Hall International, London,. ISBN 0-13-455297-0.

[3]. **A., Stankovic J.** *Misconceptions about real-time computing: a serious problem for the next-generation systems*. IEEE Computer., 1988, vol. 21, n° 10, pp 10-19.

[4]. **JEAN-PAUL CALVEZ, Jean Paul Calvez:.** *Embedded Real-Time Systems: A Specification and Design Methodology.*

[5]. **Pnueli, Z. Manna and A.** *Temporal Specification and Verification of Reactive Systems*. Vol. I and II, . s.l. : Springer, 1995.

[6]. **Systems, Wind River, Inc.** *VxWorks EVALUATION TUTORIAL* 6.7. 25 Mar 2009. Alameda, CA 94501-1153, U.S.A. Toll free (U.S.A.): 800-545-WIND.

[7]. **Systems, Wind River.** *VxWorks Programmer's Guide,* 5.4 Edition 1, 6 May 1999 Part #: DOC-12629-ZD-01.

[8]. *COMPARISON BETWEEN QNX RTOS V6.1, VXWORKS AE 1.1 AND WINDOWS CE .NET*, Dedicated Systems Experts. <*http://www.dedicated-systems.com/vpr/layout/display/pr.asp*>

[9]. **SCHWARZ J.J.; JELEMENSKA K.; ZHONGWEI HUANG.; AUBRY R.; BABAU J.-P.,** *From CRSM specification to a real-time multitasking execution model, Industrial Electronics,* 1999. ISIE apos;99. Proceedings of the IEEE International Symposium on Volume 1, Issue , 1999 Page(s):65 - 69 vol.1.

[10]. **System, Wind River.** *BSP Developer's Kit for VxWorks®User's Guide.* s.l. : , Inc,. Alameda, CA., 1999.

[11]. **Alvarez J. M., Diaz M., Llopis L., Pimentel E., Troya J. M.** *An Object-oriented Methodology for Embedded Real-time Systems.* s.l. : Computer Journal, 2003, Vol. 46, n°2,. pp. 123-145. ISSN 1460-2067.

[12**]. Luciano Lavagno, Grant Martin, Bran Selic.,***UML for real, Design of Embedded Real-Time Systems,* 2004 Springer Science,Business Media,Inc. Print 2003 Kluwer Academic Publishers, Dordrecht,. ISBN:0-306-48738-1., print ISBN: 1-4020-7501-4, p. 385.

[13]. **ALKHODRE A.; KHATAB A.; BABAU J.P.; SCHWARZ J.J.:.** *SDL and IF for Real time Systems development: specification, design and validation.* To be submitted to WRTP (accepté le 1er Juin 2004).

[14]. **R. Alur, C. Courcoubetis, and D. Dill.** *Model-checking in dense real-time. Information and Computation,* 104:2-34, 1993.

[15].**Marius BOZGA,Susanne GRAF,Luliab OBER, Laurent MOUNIER**,VERIMAG Distributed and Complex Systems <*http://www-verimag.imag.fr/people/async/if* >.

[16]. **O.M.G.: O.M.G. U.M.L. Specifications** – *Version 1.3. 1999. Disponible sur: <http://www.rational.com/uml/resources /documentation /index.jsp> (consulté le 14.06.2004) (consulté le 12.06.2004).*

[17]. **C. Joubert, R. Mateescu.** *Distributed On-the-Fly Model Checking and Test Case Generation.* s.l. : Proceedings of the 13th International SPIN Workshop on Model Checking of Software SPIN'2006 (Vienna, Austria), A. Valmari (editor), Lecture Notes in Computer Science, Springer Verlag, March–April 2006. p.126–145. vol.3925.

[18]. **Bozga M., . Fernandez J-C, Ghirvu L., Graf S., Krimm J.P. , Mounier L., Sifakis J.:.** *IF : An Intermediate Representation for SDL and its Applications.* s.l. : Proc. of the Ninth SDL Forum, Montreal, Quebec, Canada, 21-25 June, 1999,423 p.

[19]. **J.P.:, CALVEZ.** *Spécification et conception des systèmes, une méthodologie. Paris :.* Masson,1997. p. 288 . ISBN 2225822301.

[20]. **Dill., R. Alur and D.** *The theory of timed automata.* s.l. : Theoretical Computer Science,, 1994. (126):183-235.

[21]. **Gomez, Howard Bowman and Rodolfo S.** *Concurrency Theory, Calculi and Automata for Modelling Untimed and Timed Concurrent Systems.* s.l. : Springer, 2006. p.459. ISBN 1-85233-895-4.

[22]. **Katia Hristova, Yanhong A. Liu, and K. Tuncay Tekle.** *Generating Specialized Rules and Programs for Demand-Driven Analysis.* s.l. : Proceedings of the 12th international conference on Algebraic Methodology and Software Technology, AMAST 2008 (Urbana, IL, USA), Jul. 2008.

[23]. **E. Bortnik, N. Trcka, A. J. Wijs, S. P. Luttik, J. M. van de Mortel-Fronczak, J. C. M. Baeten, W. J. Fokkink, J. E. Rooda.** *Analyzing a Model of a Turntable System using Spin, CADP and UPPAAL, in:.* s.l. : Journal of Logic and Algebraic Programming, November–December 2005, vol. 65, no 2, p. 51–104.

[24]. **Kim G. Larsen, Paul Pettersson, and Wang Yi.** *Uppaal in a Nutshell. Int .* s.l. : Journal on Software Tools for Technology Transfer, 1(1–2):134–152, October 1997.

[25]. **Fernandez J-C, Garavel H., Kerbrat A., Mounier L., Mateescu R., Sighireanu M.** *.: CADP A Protocol Validation and Verification Toolbox.8th International Conference on Computer Aided Verification CAV,.* s.l. : New Brunswick (USA), LNCS 1102, 1996. pp.436-440.

[26]. **Kol, Michael Yoeli and Rakefet.** *Verification of Systems and Circuits Using LOTOS, Petri Nets, and CCS.* s.l. : Wiley, 2008. p. 231 pages. ISBN 978-0-471-70449-2.

[27]. **Schwarz J.J.** *Environnement Graphique pour La conception des Applications Multitâches Temps Réel , HDR. Lyon :.* INSA-LYON, 1995, 158 p.

[28].Cygwin User's Guide. <*http://www.cygwin.com/*>. Red Hat, Inc. [Online] [Cited: 28/ 9. 2008].

[29]. **María del Mar Gallardo, Christophe Joubert and Pedro Merino.** *On-the-Fly Data Flow Analysis based on Verification Technology.* s.l. : Proceedings of the 6th International Workshop on Compiler Optimization meets Compiler Verification COCV'2007 (Braga, Portugal), Mar. 2007.

[30]. **ALFred V.Aho, Ravi Sethi, Jeffrey D.Ullman, Monica S.Lam.:** *compilers Principles, Techniques, and tools,.* s.l. : 2nd edition PERSON Addison-Wesley publishing company, 2007. p.796 . ISBN: 0-321-48681-1.

[31]. **Levine J. R., Mason T., Brown D.** *Lex & Yacc. 2nd edition. Sebastopol, CA :.* 2nd edition. s.l. : O'Reilly, 1992. p.366. ISBN: 1565920007.

[32]. **Aaby, Anthony A.** *Compiler Construction using Flex and Bison.* s.l. : Walla Walla College 204 S. College Ave, Version of February 25, 2004. p.120.

[33]. **Wang Yi, Paul Pettersson, and Mats Daniels.** *Automatic Verification of Real-Time Communicating Systems By Constraint-Solving.*s.l. : In Dieter Hogrefe and Stefan Leue, editors, Proc. of the 7th Int. Conf. on Formal Description Techniques, North–Holland,1994. pp. 223–238.

[34]. **K. Salomaa, Sheng Yu.** *Automata Implementation***.** Springer Berlin / Heidelberg. April 11, 2006. NFA to DFA transformation for finite languages. P.149-158. ISBN: 978-3-540-63174-3.

[35]. **B. Gebremichael, T. Krilaviˇcius, and Y.S. Usenko.** *Real-time service allocation for car periphery supervision: Requirements and environment analysis,.* 2003. *<http://www.cs.kun.nl/ita/publications/papers/biniam/gku2.html>*. Internal Ametist document.

# تم بعونه تعالى

University Of Aleppo
Electricity and Electronic Engineering Faculty
Computer Engineering Department

# COMPILER DESIGN BETWEEN C LANGUAGE UNDER VXWORKS OPERATING SYSTEM AND IF (INTERMEDIATE FORMULA) LANGUAGE FOR PROGRAMS VALIDATION

Submitted in partial fulfillment of the requirements of Master Degree of Computer Engineering

**Prepared By**
**Eng. Marwa Dahdouh**

**2008-2009**